European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D6.3
Open Source Support and Joint Software Development

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
Jacobs University Bremen, JUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois - Campus scientifique
615, rue de Jardin Botanique - B.P. 101
F-54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	Open Source Support and Joint Software Development
Type:	Public
Editor(s):	Olivier Festor
E-mail:	Olivier.Festor@loria.fr
Author(s):	WP6 Partners
Doc ID:	D6.3

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.1	2007-07-03	J. Schönwälder	SMIng parser implementation
0.2	2008-02-16	F. Beck, O. Festor	NDPMon extensions
0.3	2008-02-22	D. Hausheer, M. Morariu	OpenDiameter packaging
0.4	2008-02-22	T. Bocek, F. Hecht, D. Peric	P2PFastSS implementation
0.5	2008-02-23	PSNC	Inventory data
0.6	2008-03-05	O. Festor	Integration, summary, introduction
0.7	2008-04-02	O. Festor	Quality check
0.8	2008-04-04	J. Schönwälder	ISMS implementation

Contents

1 Executive Summary

Open Source support, coherent joint software development and packaging is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster their use and acceptance on large scale.

In June 2007, two calls have been issued to the EMANICS community: one for software development and one for Open Source software packaging. Four initiatives have been selected in the first call : SMIng support in LibSMI, P2PFastSS, ISMS implementation and VoIPbots. Two proposals have been selected in the packaging part: Open Diameter packaging and NDPMon distribution standardization and integration in standard distributions. In addition, the work on Management software categorisation, annotation and referencing did continue.

This report provides the details of the two calls that have been issued since June 2007 and for every selected project, a detailed description of the software package together with the achievements made so far are presented. The work on the software inventory database is also reported in this document.

A revised version of this document is expected to be provided at the end of phase 2 of EMANICS in December 2008.

2 Introduction

Open Source support, coherent joint software development and packaging is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster their use and acceptance on large scale.

This effort was already initiated in the first phase of the network of excellence. The results obtained there are are reported in deliverable D6.2. In this report, we cover the activities undertaken from july 2007 to march 2008. A revised version of this deliverable is planed for 12/2008.

In June 2007, two calls have been issued to the EMANICS community: one for software development and one for Open Source software packaging. Four initiatives have been selected in the first call : SMIng support in LibSMI, P2PFastSS, ISMS implementation and VoIPbots. All except one of these selected projects did start immediately after selection. Only the VoIPbot project was delayed at INRIA replaced by NDPMon extensions documented in this report.

Two proposals have been selected in the packaging part: Open Diameter packaging and NDPMon distribution standardization and integration in standard distributions. In addition, the work on Management software categorisation, annotation and referencing did continue.

To present the achievements, the report is structured as follows. Section 3 is dedicated to the LibSMI development for SMIng support, first selected development project in phase 2 of EMANICS. Section 4 contains the description of both the package and the new developments undertaken in phase 2 of EMANICS on NDPMon. Section 5 contains the ISMS development report. Section 6 presents the developments realized in the P2PFastSS similarity search funded initiative. Sections 7 and 8are dedicated to the two selected packaging initiatives. Section 9 provides the description of enhancements brought to the software inventory and search service offered within EMANICS. Section 10 concludes this deliverable. The text of the first two calls issued in July 2007 is given as an appendix to this document.

3 SMIng Parser Implementation

The `libsmi` library is an open source C library for parsing data models written in the SMI/SPPI languages. The library comes with a collection of tools to validate data models, to convert data models into various formats, and to analyze changes made between different versions of a data model for backwards compatibility. The library and the associated tools are widely using within the industry and for validation purposes by standardization bodies such as the IETF. The aim of this project was to add support for SMIng [1] to the `libsmi` library.

3.1 Presentation

The Structure of Management Information (SMI) is the language for describing Internet Management Information Bases (MIBs). There are two versions of the language that are in use: SMIv1 [2] and SMIv2 [3, 4, 5]. SMIv2 has superseded SMIv1 as the language used for MIB definitions, and most vendors use it even if they ship SMIv1 definitions to customers. This transition is supported by open source tools such as `libsmi`, which provides an API for accessing SMI structures and provides the tools for automatic conversion between the different SMI languages.

There are several problems with SMIv2. One of the main problems is its dependence on the ASN.1 syntax which is controlled by the ISO organization and not by the IETF. Furthermore the dependence is on an old version of ASN.1, which leads to many difficulties for developers who need more information on the syntactical rules of the [6]. The main difficulty is the implementation of parsers, which cannot rely on a BNF (Backus-Naur Form) grammar, which is the usual way for translation of language syntax to lexical and syntactical rules for parser generators. This leads to many developers using “fuzzy” MIB scanners that do not check for conformance with the SMI syntax [6]. Another problem with SMIv2 is that its syntax does not allow for efficient parsing, which causes many vendors to use more efficient but proprietary intermediate formats, which harms interoperability [6].

SMIng [1] is a new data definition language that addresses the problems of SMIv2. It is independent from other language definitions and is instead defined using ABNF grammar, which greatly simplifies the implementation of parsers [6]. Additionally SMIng syntax is greatly simplified because of a clear separation of statements by semicolon, which makes error recovery in parsers much easier [6]. It includes a more complete set of base data types. Its syntax is arguably simpler to read even for an experienced programmer that does not necessarily have experience with SMIng.

Further advantage of SMIng is its protocol independence. SMIng definitions are not tied to a specific network management protocol, so SMIng extensions, that define an ABNF grammar for conversion, are used for the actual mapping of SMIng definition to different concrete protocol implementations, such as SNMP [7] or COPS-PR [8] or NETCONF [9]. Another SMIng advantage are the language extensions, as that they will allow old parsers to work with newer definitions, should new syntax rules be added to the language [6].

The main statements in an SMIng module are: extensions, identities, type definitions (typedefs), and classes. Extensions provide mechanism for extending the language syntax, and to map its syntax to other languages. Identity statements define abstract and

untyped identities, and describe their semantics. They can be derived from another identity. Typedefs derive new data types by restricting the base types, or other derived types. The base types of SMIng are: 32 and 64-bit signed and unsigned integers; 32, 64, and 128-bit floating point numbers; enumeration type, that provides named numbers; named bits type for storing binary flags; object identifier type, used to describe the path in the registration tree for SMIv1/v2 nodes; and a pointer base type that is used to reference class or identity [1].

SMIng classes represent a container of related attributes and events. Attributes are either data types, or other class references. Events that represent asynchronous events related to the event statement's class. Classes can be derived from another class [1]

Despite its advantages, SMIng is not yet widely used. One reason is likely the lack of convenient programming tools to work with SMIng definitions. In particular, one of the obstacles to adoption of SMIng is the lack of an SMIng parser that provides a convenient API to access SMIng structures.

A potential tool for that task is `libsmi`, which is a portable C library that currently provides such an API for access to SMIv1 and SMIv2 definitions. This library hides parsing details providing accessing SMI definitions for applications that among many useful tasks can be used to translate between the different SMI versions.

The purpose of this project was to update `libsmi` to support the SMIng language. This included providing a lexical analyzer and parser for the language, updating the API to support iterators for the new features of SMIng and providing a dump tool that can be used to test the implementation, and to save module information from memory to the usual textual form of SMIng module.

The main goal of this library update is to create a useful tool on which future SMIng tools can be build and by that to promote adoption of SMIng.

3.2 Libsmi Library

The `libsmi` library provides an API for accessing SMI module structures like modules definitions, type definitions, registered nodes in a registration tree, etc. The access can be done by iterators and by direct access to names or object identifiers. The structure of the library is two-layered (see Figure 1). The upper layer provides data structures to management information in memory and implements the API. The lower layer or the backends are the actual parsers that retrieve the management information from the SMI definitions. There are three currently available language back-ends for SMIv1 and SMIv2 and SPPI. There was an incomplete backend based on early version of SMIng, which was very different from the current language definition.

The goal of the project was to add support for the SMIng language as defined in [1], and to extend the `smidump` tool that comes with the library to dump SMIng modules.

3.3 Implementation Progress Report

The execution of the project started by implementing a lexical analyzer in `flex`. Apart from tokenizing keywords, identifiers, text segments, and number definitions, the `flex` scanner

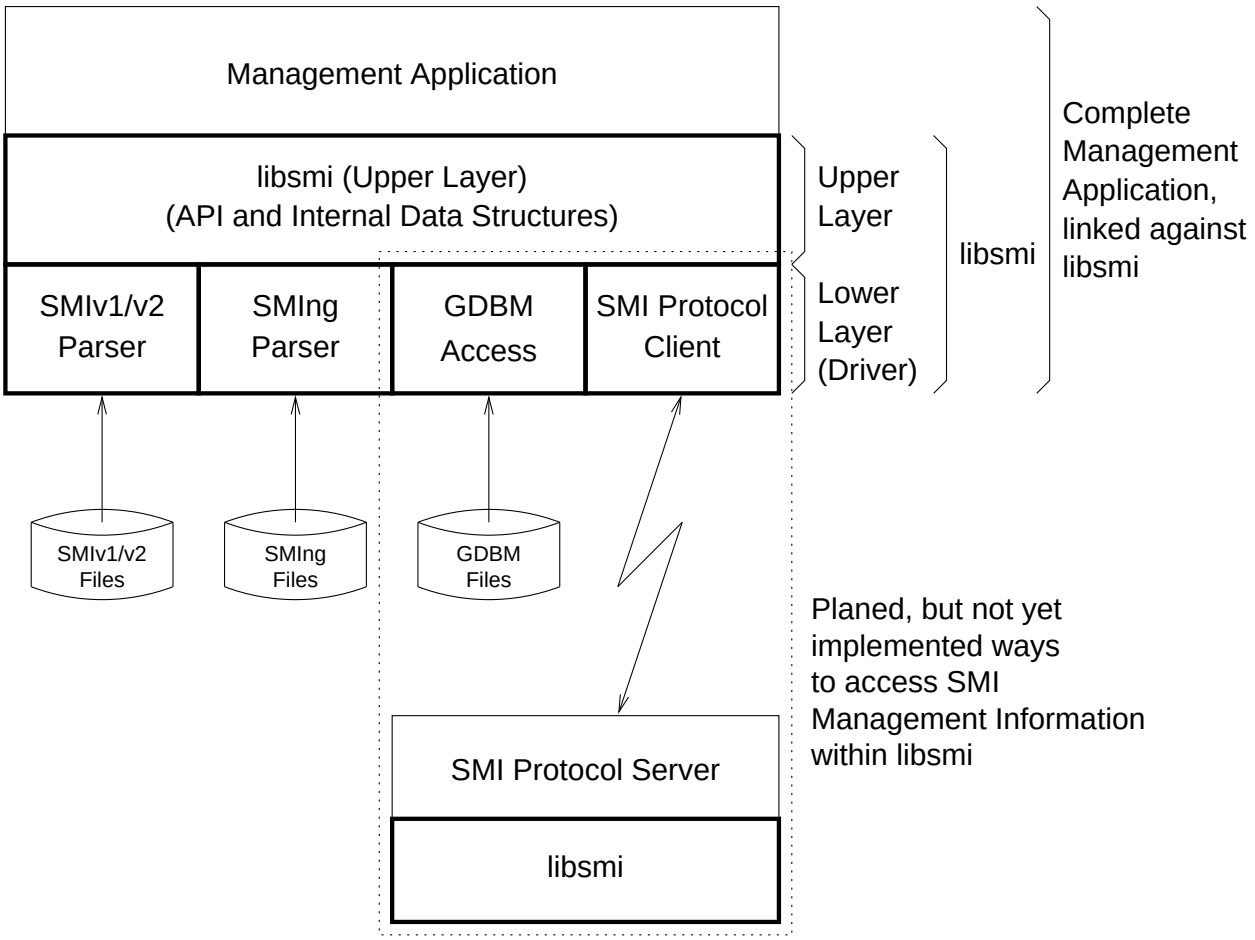

Figure 1: Internal structure of the `libsmi` library

deals with removing comments and indentation from text segments, and checking of length constraints for identifiers.

The next level of parsing is done by a Gnu `bison` parser. It implements the SMIng ABNF grammar, with minor additions, to make some of the rules better suited for LR parsing. A major difference with the definition of SMIng in [1] with the implementation is the change of the placement of identity statements. Originally they were supposed to be placed after type definitions and before class definitions. The implementation changed identity statements to be before type definitions. The motivation behind this decision was to remove the forward references, which is one of the goals of SMIng. The problem with forward referencing came up because Pointer values in type definitions reference identities, so identities should be defined before the type definitions that reference them.

The first element of the SMIng definition that is read is the module and its import statements. If import statements are present, the corresponding modules are loaded into memory. For any statement that uses identifiers, a corresponding find function (like `findType()`, `findIdentity()`, `findClass()`, etc.) is used to check the identifiers for correct usage, that is to check whether such identifier is previously defined in case of a reference, or to check for repetition of names. If the referenced identifier is no found, or if it is already used for the same type of statement when defining new statement, the parser prints an error. The implementation, first searches in the current module for the corresponding statement

```
typedef struct SmiType {
    SmiIdentifier       name;
    SmiBasetype         basetype;
    SmiDecl             decl;
    char                *format;
    SmiValue            value;
    char                *units;
    SmiStatus           status;
    char                *description;
    char                *reference;
} SmiType;
```

Figure 2: External representation of a type in `libsmi`

```
typedef struct Type {
    SmiType         export;
    Module          *modulePtr;
    struct Type     *parentPtr;
    struct List     *listPtr;
    TypeFlags       flags;
    struct Type     *nextPtr;
    struct Type     *prevPtr;
    int             line;
} Type;
```

Figure 3: Internal representation of a type in `libsmi`

definition, then searches among the imported identifiers, and if an imported identifier with the same name is found, the corresponding module is searched for the corresponding statement definition.

3.3.1 Data Structures

The output of the parser is stored in memory in data structures internal to `libsmi`. The data structures also have two layers: public and private. Most of the API functions return structures that contain the data of a given statement and are named with an smi prefix, for example `SmiType`, `SmiIdentity`, `SmiModule`, etc. These structures contain the data for the given statements and include the description, status, reference, and other data fields specific for the type of statement. For example the `SmiType` structure contains a field for a default value. The private part of the data structures holds lists of the corresponding smi structures, and some data that is meant to be accessed through functions, such as parent types, modules associated with the current statement and others. The private structures all are named after their smi-prefixed counterparts. A representative example is `SmiType` shown in Figure 2 and the private structure `Type` shown in Figure 3, which contains a member `struct SmiType export`.

```
typedef struct SmiIdentity {
    SmiIdentifier       name;
    SmiDecl             decl;
    SmiStatus           status;
    char                *description;
    char                *reference;
} SmiIdentity;
```

Figure 4: External representation of an SMIng identity in `libsmi`

```
typedef struct SmiClass {
    SmiIdentifier       name;
    SmiDecl             decl;
    SmiStatus           status;
    char                *description;
    char                *reference;
} SmiClass;
```

Figure 5: External representation of an SMIng class in `libsmi`

As SMIv1, SMIv2, and SMIng are used to describe similar data, there are some similarities in the logic behind the languages which leads to the possibility of reuse of the internal data structures, and to transcode SMIv2 modules to SMIng and vice-versa, at least for modules that contain only type and extension definitions. However, for full conversion of SMI modules, one needs SMIng extensions to define the complete SMIng mapping to SMIv2 [10]. The conversion between SMI languages was beyond the scope of this project.

The data structures that were reused were the `SmiType`, `SmiModule`, `SmiImport`, and `SmiMacro`. The `SmiModule`, `SmiType`, and `SmiImport` structures were unchanged from the last version of the library. The `SmiMacro` struct, which holds SMIng extensions, was extended with an `abnf` string member, to hold the ABNF information of SMIng extension. The `SmiValue` value union, which holds default values for `SmiType`, was updated with the floating point number types.

The public structures contain fields of enumerated C types, which define the different values this fields can hold. Several of this enumerated types were extended with new definitions specific to SMIng. The SmiBasetype was extended with floating point values `SMI_BASETYPE_FLOAT32`, `SMI_BASETYPE_FLOAT64`, `SMI_BASETYPE_FLOAT128`, and the new SMIng pointer base type `SMI_BASETYPE_POINTER`. The `SmiAccess` enum was extended with the SMIng `SMI_ACCCESS_EVENT_ONLY` access mode. `SMI_DECL_IDENTITY`, `SMI_DECL_CLASS`, `SMI_DECL_ATTRIBUTE`, and `SMI_DECL_EVENT` were added to the `SmiDecl` enumeration that defines type of statements and can be used to retrieve definitions.

The new public structures that were added were `SmiEvent` representing class events (Figure 7), `SmiAttribute` representing class attributes (Figure 6), `SmiClass` representing classes (Figure 5) and `SmiIdentity` for SMIng identity definitions (Figure 4). Several private structures were added corresponding to the new public structures as shown in Figures 8, 9, 10, and 11.

```
typedef struct SmiAttribute {
    SmiIdentifier       name;
    SmiBasetype         basetype;
    SmiDecl             decl;
    char                *format;
    SmiValue            value;
    char                *units;
    SmiStatus           status;
    char                *description;
    char                *reference;
    SmiAccess           access;
} SmiAttribute;
```

Figure 6: External representation of an SMIng attribute in `libsmi`

```
typedef struct SmiEvent {
    SmiIdentifier       name;
    SmiDecl             decl;
    SmiStatus           status;
    char                *description;
    char                *reference;
} SmiEvent;
```

Figure 7: External representation of an SMIng event in `libsmi`

```
typedef struct Identity {
    SmiIdentity         export;
    Module              *modulePtr;
    struct Identity     *parentPtr;
    struct Identity     *nextPtr;
    struct Identity     *prevPtr;
    int                 line;
} Identity;
```

Figure 8: Internal representation of an SMIng identity in `libsmi`

```
typedef struct Class {
    SmiClass            export;
    Module              *modulePtr;
    struct Attribute    *firstAttributePtr;
    struct Attribute    *lastAttributePtr;
    struct List         *uniqueList;
    struct Event        *firstEventPtr;
    struct Event        *lastEventPtr;
    struct Class        *parentPtr;
    struct Class        *nextPtr;
    struct Class        *prevPtr;
    int                 line;
} Class;
```

Figure 9: Internal representation of an SMIng class in `libsmi`

```
typedef struct Attribute {
    SmiAttribute        export;
    Class               *classPtr;
    struct Type         *parentTypePtr;
    struct List         *listPtr;
    struct Attribute    *nextPtr;
    struct Attribute    *prevPtr;
    int                 line;
    struct Class        *parentClassPtr;
} Attribute;
```

Figure 10: Internal representation of an SMIng attribute in `libsmi`

```
typedef struct Event {
    SmiEvent            export;
    Class               *classPtr;
    struct Event        *nextPtr;
    struct Event        *prevPtr;
    int                 line;
} Event;
```

Figure 11: Internal representation of an SMIng event in `libsmi`

All the private structures are elements of doubly-linked lists and have references to the previous and the next element. All of them have as first element a corresponding `libsmi` public structure. This is used by internal library functions to cast public structure pointers to private structure pointers. The `Class` and `Identity` structures have pointers to their module and to the parent class or identity from which they are derived. The `Attribute` and `Event` structures do not have a module pointer, but instead have a `Class` pointer, as they are tied to their class and not directly to a module. Each `Attribute` has either a parent class or parent type specified through the corresponding pointer. The `Class` has pointers to a list of the `Attribute` structures that it contains and to a list of unique elements. The list is either empty (`uniqueList=NULL`) if the class is not meant to be instantiated separately, or the list's first element points to the current class (`Class->uniqueList.ptr = Class`) if the class is scalar, that is its unique list is empty and it is meant to have only one instance, or contains a list of `Attribute` structures, that uniquely defines the class instance.

3.3.2 Functions

The private API of the library was extended with setter functions like `setClassName()`, `setClassDecl()`, `addClass()` and other similar methods for the `Identity`, `Event`, and `Attribute` structures. The `Event` and `Attribute` functions differ from `Identity` and `Class` functions in the fact that for example `addClass()` function takes as one of its arguments `Module` pointer, whereas `Event` and `Attribute` functions take as argument, a `Class` pointer, because they are tied to their class, and not directly to the module. The `Attribute` structure also has a `duplicateTypeToAttribute()` function, that duplicates an existing `Type` to an `Attribute`.

The public API was extended with the corresponding `smiGet*()` iterators, for the `Identity`, `Class`, `Event`, and `Attribute` structures:

- The function `smiGetFirstIdentity(SmiModule *smiModulePtr)` together with the function `smiGetNextIdentity(SmiIdentity *smiIdentityPtr)` are used to iterate through the identities of the module given by `smiModulePtr`. They return a pointer to `struct SmiIdentity` that represents an identity or `NULL` if there are no identities left in the module, or an error has occurred.
- The function `smiGetIdentity(SmiModule *smiModulePtr, char *name)` returns a pointer to the `struct SmiIdentity` for the identity with the given name in the given module, or `NULL` if the idenity with the given name does not exist.
- The function `smiGetIdentityModule(SmiIdentity *smiIdentityPtr)` returns a pointer to `struct SmiModule` of the module containing the given identity.
- The function `smiGetParentIdentity(SmiIdentity *smiIdentityPtr)` returns a pointer to a `struct SmiIdentity` pointing to the parent of the given `smiIdentityPtr`, or `NULL` if the idenity is not derived.
- The function `smiGetFirstClass(SmiModule *smiModulePtr)` together with the function `smiGetNextClass(SmiClass *smiClassPtr)` are used to iterate through all the classes of the module given by `smiModulePtr`. They return a pointer to `struct`

SmiClass that represents a class or NULL if there are no classes left in the module, or an error has occurred.

- The function smiGetClass(SmiModule *smiModulePtr, char *name) returns a pointer to struct SmiClass that represents the class with the given name in the current module, or NULL if the class with the given name does not exist.
- The function smiGetClassModule(SmiClass *smiClassPtr) returns a pointer to struct SmiModule of the module containing the given class.
- The function smiGetParentClass(SmiClass *smiClassPtr) returns a pointer to struct SmiClass pointing to the parent of the given smiClassPtr, or NULL if the class is not derived.
- The function smiIsClassScalar(SmiClass *smiClassPtr) returns int 1 if the class is a scalar (its unique statement contains an empty list) or 0 otherwise. This function can be used in conjunction with smiGetFirstUniqueAttribute() to determine whether the class is meant to be instantiated separately (has unique statement with nonempty list), or if it is meant to be used as part of another class (has no unique statement).

Some of the fields of an SmiAttribute may be empty if it references a class instead of a type. There are getAttributeParent*() functions for retrieving attribute parent, type or class. They can be used to determine if an attribute references a type or class. Also functions for retrieving named numbers and ranges of SmiAttribute's, were added.

- The function smiGetAttribute(SmiClass *smiClassPtr, char *name) returns a pointer to the struct SmiAttribute for the attribute with the given name in the current module, or NULL if the attribute with the given name does not exist.
- The function smiGetFirstAttribute(SmiClass *smiClassPtr) together with the function smiGetNextAttribute(SmiAttribute *smiAttributePtr) are used to iterate through the attributes of the class given by smiClassPtr. They return a pointer to a struct SmiAttribute that represents an attribute or NULL, if there are no attributes left in the class, or an error has occurred.
- The function smiGetAttributeParentClass(SmiAttribute *smiAttributePtr) returns pointer to a struct SmiClass pointing to the parent of the attribute identified by smiAttributePtr, or NULL if the attribute does not reference class. Note that attributes always have either parent type or parent class.
- The function smiGetAttributeParentType(SmiType *smiAttributePtr) returns a pointer to a struct SmiType pointing to the parent type of the given smiAttributePtr, or NULL if the attribute does not reference type. Note that attributes always have either parent type or parent class. The functions smiGetAttributeParentClass() and smiGetAttributeParentType() can be used do determine what type reference, class or type, the given attribute is.

- The function `smiGetFirstUniqueAttribute(SmiClass *smiClassPtr)` together with the function `smiGetNextUniqueAttribute(SmiType *smiAttributePtr)` are used to iterate through the unique attributes of the module given by `smiClassPtr`. They return a pointer to `struct SmiAttribute` that represents a unique attribute or `NULL` if there are no unique attributes left in the class, or an error has occurred. This function MUST NOT be used for scalar classes, so it should only be called after isClassScalar() has returned 0.

- The function `smiGetEvent(SmiClass *smiClassPtr, char *name)` returns a pointer to the `struct SmiEvent` for the attribute with the given name in the current module, or `NULL` if the event with the given name does not exist.

- The function `smiGetFirstEvent(SmiClass *smiClassPtr)` together with the function `smiGetNextEvent(SmiEvent *smiEventPtr)` are used to iterate through the events of the class given by `smiClassPtr`. They return a pointer to `struct SmiEvent` that represents an event or `NULL` if there are no events left in the class, or an error has occurred.

- The functions `smiGetAttributeFirstRange(SmiAttribute *smiAttributePtr)` and `smiGetAttributeNextRange(SmiRange *smiRangePtr)` are used to iterate through ranges that restrict number or octet string types. Both functions return a pointer to the `struct SmiRange` representing the range, or `NULL` if there are no more ranges, or an error has occurred.

- The functions `smiGetAttributeFirstNamedNumber(SmiAttribute *smiAttributePtr)` and `smiGetAttributeNextNamedNumber(SmiNamedNumber *smiNamedNumberPtr)` are used to iterate through named numbers of bits or enumerations for attributes, which reference types, and to retrieve the reference restriction of a pointer. Both functions return a pointer to the `struct SmiNamedNumber` representing the named number, or `NULL` if there are no named numbers left, or an error has occurred. The function `smiGetFirstNamedNumber()` can be used to retrieve the name of the identity that is restricting a pointer type, as it is stored as the name of the first named number.

3.3.3 Smidump driver

The final part of the project was to update the dump tool `smidump`, which is part of the `libsmi` package. The `smidump` tool has several drivers for dumping SMI structures into different formats. As part of the EMANICS funded development, the SMIng driver was updated to print the memory structures of `libsmi` back to text format according to the SMIng module definitions. The tool's most important task within the project was that it was used for testing of the parser. The driver prints the data structures information and the corresponding keywords, that would form a valid module, semantically identical to the the input module.

3.4 Testing

Initial testing was done manually by writing short modules that contains single, or in case of referencing tests several, statements, and checking whether the retrieval functions (the `smiGet*()` functions), return structures with the same information as the input. This was done to decouple the testing of the parser from the testing of `smidump`.

Several modules were written, to represent most of the statement combinations that occur in SMIng. More specifically typedefs for all the base types were written, and checked whether the internally stored information matches the initial module definition. Additionally derived types were generated to check whether derivation of types works correctly. The same was done for identities, extensions, and classes. Classes' attribute and event statements were also tested for correctness.

Consistency tests were performed for all types of statements that are defined by referencing. They were checked with modules that have incorrect references (for example a typedef that derives from not yet defined type, or an identity that is derived from an undefined or non-imported parent identity). Tests with modules that have redefinition of identifiers were performed as well. The parser raises errors in all cases where inconsistent reference or redefined name exists.

After the parser was checked component-wise, an SMIng driver was written for `smidump` and several larger modules comprising of combination of the smaller test cases were tested with `smidump`. The output of the dump tool was manually checked with the help of `diff` for differences between the output module and the original module, and no semantic differences were found. Some other modules that were used for testing the parser were the `IRTF-NMRG-SMING` modules that are distributed as part of `libsmi` MIB module collection.

3.5 Conclusions

Although the parser works, as expected, no real-world users have tested it so far, which means that there might be yet unknown bugs, or features perceived as bugs. In order to ensure the quality of the new functionality of `libsmi` added by this project, the SMIng parser will need to gain some user base. Although the API includes all the essential functions for retrieving information from SMIng modules, some usage patterns might not be known at the moment, and future feature requests to address user needs are possible.

The SMIng language has not yet gained industrial application, but it's advantages are numerous and recognized in the management community. The `libsmi` library, with its SMIng parser, is a very useful tool, providing an easy way to write programs that use SMIng definitions. However, there is room for future improvement of the library. The first important and needed improvement is the update of the drivers of `smidump` adding SMIng support. Another important improvement is the creation of a tool that can convert SMIv1/v2 modules to SMIng modules, and vice versa, which is essential for transition from older MIB implementations and compatibility with them. This feature might be integrated into dump drivers, as a separate application, or as a feature of the `libsmi` library itself. Despite the needed improvements, `libsmi` already provides good base for creation of SMIng capable applications.

4 NDPMon extensions

4.1 Presentation

NDPMon is Open Source software for IPv6 Neighbor Discovery Protocol monitoring. It is distributed under the LGPL licence as a SourceForge at the URL `http://ndpmon.sf.net`. Initially designed with a grant from CISCO Systems in 2006, the development of the tool was pursued with EMANCIS funding in 2007 were it first went public. Since a couple of extensions and portings have been realized. They are reported in this section.

Overview

ArpWatch (`http://ee.lbl.gov`) is used in many networks and is well known by network administrators as one of the core components of the monitoring plane for IPv4 networks. Its role is to monitor the activity of the ARP protocol. By analyzing ARP packets, it maintains an up-to-date cache in which the pairing between IPv4 and Ethernet addresses for all hosts on the link are stored. A timestamp is associated to each entry in this cache, which enables a monitoring over the time of host activity. When an ARP packet is captured, it is compared to the information in the cache, and if a suspicious behavior, or special activities (a new station has appeared in the network) are detected, a report is sent to the administrator. These reports are sent via various channels including syslog and mail for the most common ones.

NDPMon, Neighbor Discovery Protocol Monitor, is an IPv6 version of ArpWatch. It initially performs the same monitoring tasks as ArpWatch, but for IPv6. It is in charge of monitoring the Neighbor Discovery Protocol activities and maintains a neighbor database up-to-date, which contains the correspondences between IPv6 and Ethernet addresses, alongside with a Time-stamp. When a Neighbor Discovery packet is captured, the content is compared to the entries in the database. Usually, in IPv4, only one address was assigned to an interface. In IPv6, multihoming is one of the key features, for Network Renumbering [11] for example. When defining the neighbors cache entries, we have to take this specificity into account. In the same way than ArpWatch, activities and suspicious behaviors raise alerts and reports. All these alerts are sysloged, and depending on the syslog daemon configuration, they can be sent to a remote station. The most severe ones, namely the suspicious activities, raise alerts, which are sent by mail to a defined address, by using an external Mail Transfer Agent (MTA).

In addition to this monitoring role, NDPMon is also able to detect attacks against the Neighbor Discovery Protocol, as defined in [12], misconfiguration, stack vulnerabilities and suspicious behaviors.

Architecture

The tool is designed to be deployed on each subnet or link of the network, as shown in figure 12. It operates in two phases: a learning phase and a monitoring phase. During

the learning phase, NDPMon builds the neighbors database by capturing the Neighbor Discovery messages. During this phase, the tool does not send any report. NDPMon makes the assumption that, when it enters the learning phase, the network is healthy and that all the activities are legitimate.

Once this phase is over, the tool can switch to monitoring mode. Thus, based on the Router Advertisements received, it also populates the routers list. Switching can be done either remotely or by restarting the daemon with the right parameters.

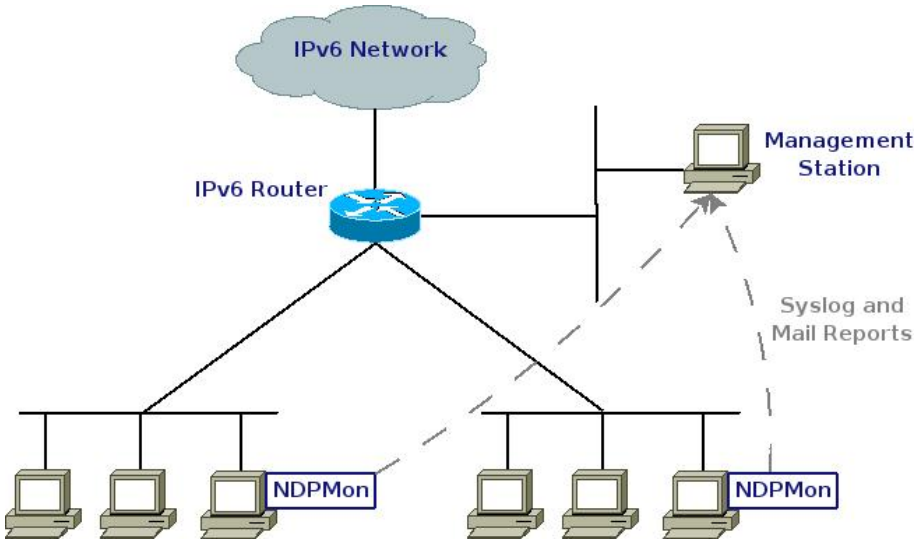

Figure 12: NDPMon Deployment

The software that implements NDPMon has been coded in the C language. It must be launched with root privileges and runs as a daemon. To limit as much as possible the resource consumption, we performed some code profiling with the tool ValGrind (`http://valgrind.org/`). NDPMon uses two XML files, which are parsed and modified by libxml2 (`http://xmlsoft.org/`) (see Figure 13).

The first file is the configuration file, which contains two main parts. First, there are some options for the configuration of the daemon itself: syslog facility to use, Email address for the reports... The administrator must also give the list of the legitimate routers on the link (Ethernet and IPv6 addresses, advertised IPv6 prefixes). This information is used by the daemon as the legitimate routing data on the link, and is used as references when Neighbor Discovery packets are received. If the tool is launched in learning phase, the information related to the authorized routers is completed with the information contained in the received Router Advertisements.

The second file is the neighbor cache. It contains the pairings between IPv6 and Ethernet addresses. A time-stamp is set for each entry in the cache. This file is filled automatically by the daemon. If NDPMon runs in learning phase, this database will be constructed without sending any report.

When the tool is launched, these two files are parsed. The neighbor cache is loaded and the entries are converted in C structures. The routing information in the configuration file is converted in the same way. The daemon begins to listen to the network and captures

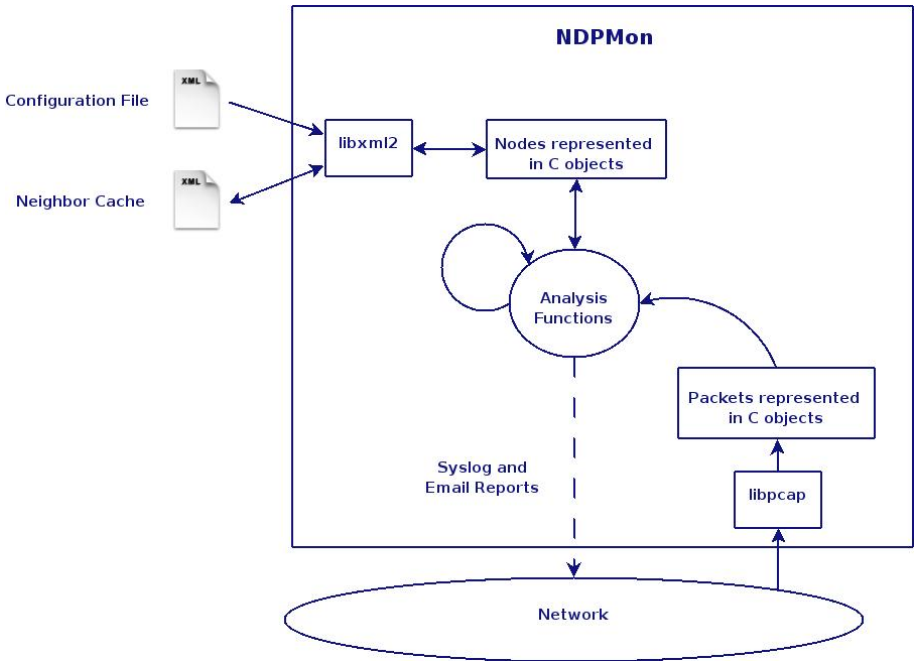

Figure 13: NDPMon Implementation Architecture

Neighbor Discovery packets thanks to the libpcap (`http://www.tcpdump.org/`). These packets are converted in C structures. The tool implements analysis functions to verify if the packet is legitimate, or if it is a malicious packet.

If the packet is considered malicious, reports are sent via syslog, and, depending on the severity, a mail is sent to the administrator. If the packet is legitimate, the information it contains is added in the neighbor cache if they are not already present; otherwise the Time-stamp of the corresponding is simply updated.

Monitored Activities

The Neighbor Discovery activities monitored by the tool are:

- New station: a new Ethernet address appears on the network, a new node has appeared
- New IPv6 Global Address: a new IPv6 global address is detected for a host
- New Link Local Address: a new IPv6 link local address is detected for a host
- New activity: the source node had no activity during the last month
- Bogon: the IPv6 source address is not local to the link
- Ethernet mismatch: the Ethernet address specified in the ICMP option is not the same than the Ethernet source address of the packet

- Changed ethernet address: a node changed its Ethernet address while keeping the same IPv6 address
- Flip flop: a node is switching between two different Ethernet addresses
- Reused address: a node is re-using an old Ethernet address

There are several possible attacks against the Neighbor Discovery Protocol. Monitoring only the classic Neighbor Discovery activities is not sufficient for detecting them. To detect these attacks we must, besides the neighbors cache, also define the legitimate behavior for the routing on the link. By defining the router Ethernet and IPv6 valid addresses, and the legitimate prefixes on the link, we can detect the following attacks:

- Wrong MAC/IP pair: Separately, the MAC and IP addresses are valid, but not as a couple
- Unknown MAC Manufacturer: the vendors of the MAC address is not known, it may be forged
- Wrong router mac: the Ethernet source address of the Router Advertisement is not defined as valid
- Wrong router ip: the IPv6 source address of the Router Advertisement is not defined as valid
- Wrong prefix: the prefix advertised in the Router Advertisement is not legitimate on the link
- Wrong router redirect: the source of the Redirect message is not a legitimate router
- NA router flag: the Neighbor Advertisement has the Router flag set whereas it is not defined in the known routers list
- DAD DoS: Denial of Service toward the Duplicate Address Detection mechanism
- Ethernet broadcast: the Ethernet source address is the broadcast address
- IP multicast: the IPv6 source address is a specific multicast address

When detected, all these behaviors trigger the sending of syslog and mails alerts. Some of them are not only the manifestation of a malicious behavior or a misconfiguration; they symbolize vulnerabilities in the IPv6 stack. For example, depending on the version of the Linux kernel, and thus of the IPv6 stack, it is possible or not to assign an Ethernet broadcast address on an interface. If such a behavior is detected, it means that the host on which this address is set has an old version of the IPv6 stack, and that it could be judicious to update it.

4.2 Expected Impact

The software is meant to be deployed in all IPv6 networks to help administrators to monitor the nodes activity, detect unexpected behaviors or presence of nodes on their network, and collect some statistics about it. We would like to deploy it on as many networks as possible, to gather as much feedback and experiences as possible.

4.3 Progress Report

In this section, we will present the progresses and new features brought to NDPMon since the beginning of the EMANICS support campaign in 2007.

New Alerts

By taking as references the RFC2461 (IPv6 Neighbor Discovery Protocol [13]) and RFC2462 (IPv6 Stateless Address Autoconfiguration [14]), we identified new threats and implemented the associated attacks:

- Wrong ipv6 router: if neither the Link Local Address and the MAC address are known for a RA
- Wrong RA flags: if the managed and other flags in the RA are not well set
- Wrong source link address option: the MAC address in the Link Adress option does not match with the Ethernet source address
- Wrong ipv6 hop limit: IPv6 Hop Limit is not 255
- Wrong RA lifetimes: preferred lifetime is bigger than the valid lifetime
- RA valid lifetime too short: valid lifetime is less than 2 hours

These new alerts are integrated in version 1.3c dated from the 25 of October 2007.

Multi-OS Porting

NDPMon has been ported under the following new Operating Systems:

- FreeBSD - release 1.2 31 July 2007
- OpenBSD release 1.2a - 28 August 2007
- NetBSD release 1.3 - 10 September 2007
- Mac OS X 10.4 Tiger release 1.3 - 10 September 2007

GUI

Alongside with release 1.3a from the 25 of September 2007, we implemented and released a User Interface for NDPMon. It is actually a WEB interface. The daemon has been modified as follows:

- All alerts are piped through the Python script create_html_table.py into an XML file alerts.xml
- Each time the neighbors cache is written, the daemon stores the number of neighbors currently discovered with a timestamp in the file discovery_history.dat

These two files are used by the WEB interface, the first one to display the alerts and reports in an HTML table with colours codes (green is for information, orange for warning, and red for problem), as shown in figure 14, and the second one to generate statistics over the discovery.

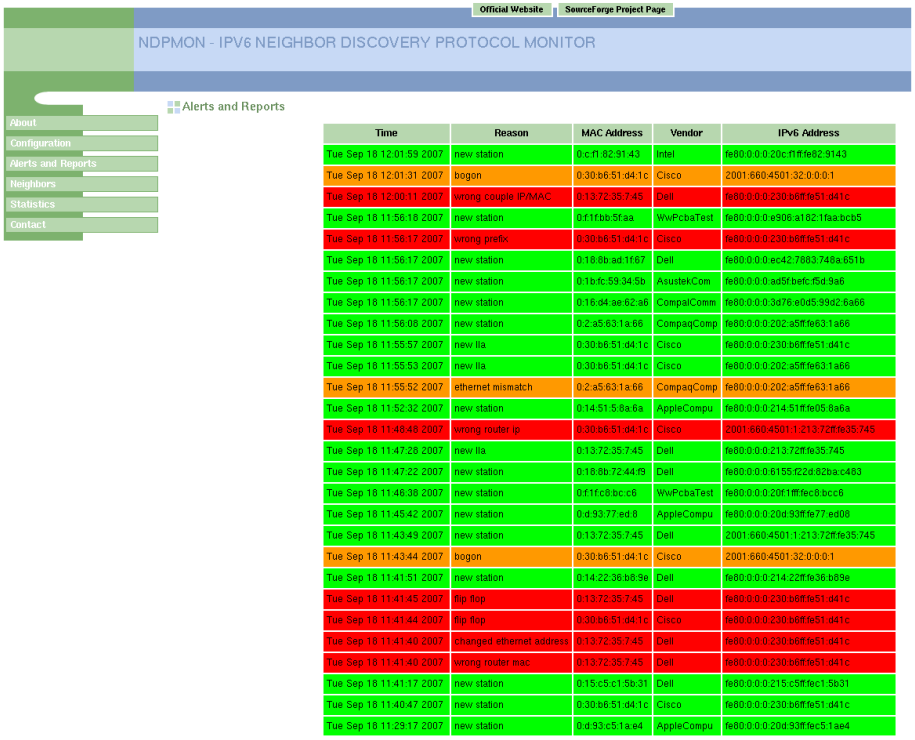

Figure 14: NDPMon WEB Interface - Alerts

All information contained in the XML files (configuration, alerts and neighbors) is displayed in HTML tables via XSL transformations. The WEB interface generates statistics over the discovered nodes and the Mac Manufacturers as shown in figure 15.

4.4 Conclusion

The successive support intitatives of EMANCIS have enabled the maturation of the NDPMon framework. The tool is now complete, tested and operational for large Ipv6 networks.

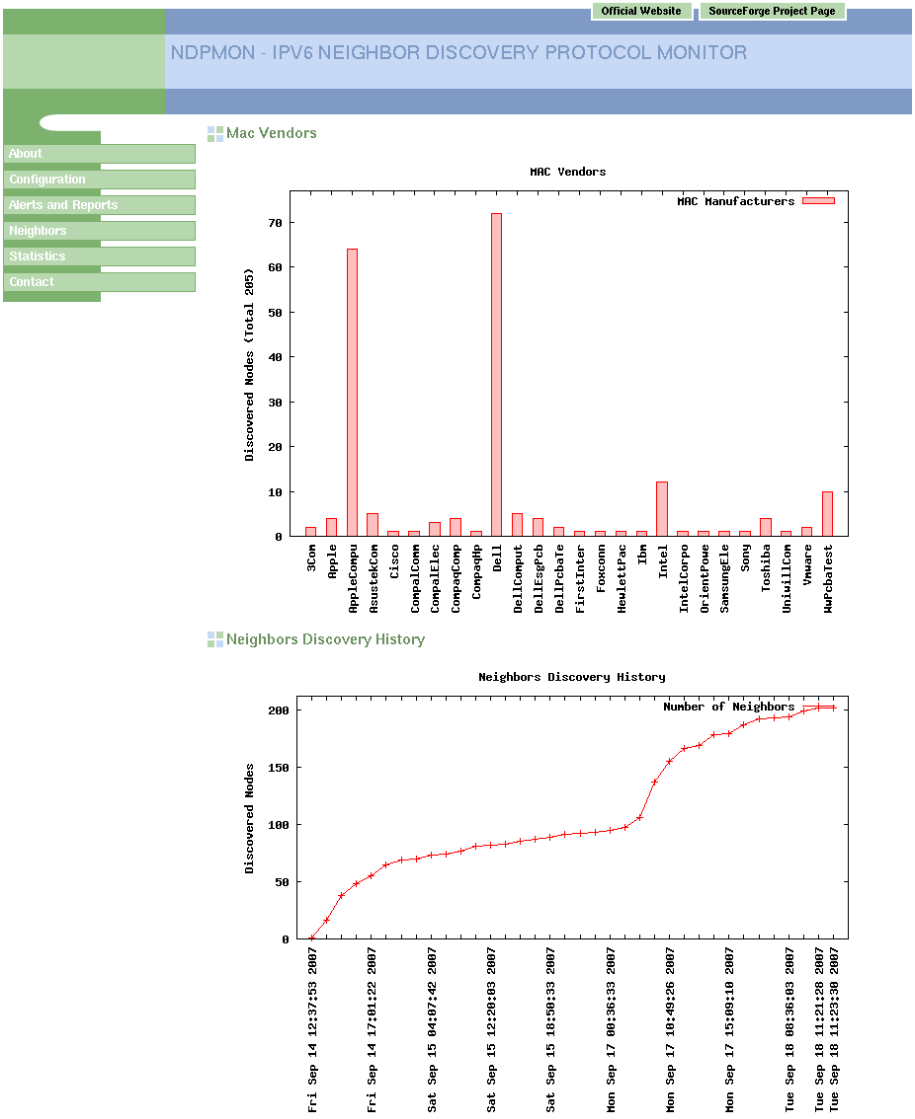

Figure 15: NDPMon WEB Interface - Statistics

Its full-fledged graphical user interface and multi-OS availability are strong elements for its acceptance. We currently witness an increase in the community around the framework and are aware of several large scale real deployments which is very encouraging.

5 ISMS Implementation

The Integrated Security Model for SNMP working group (ISMS) of the Internet Engineering Task Force (IETF) is working on specifications that extend the SNMP architecture [15] with a transport subsystem [16] and a transport security model [17] so that SNMP can run over secure transport protocols, such as SSH or TLS. The ISMS working group selected SSH [18] as a secure transport required to implement [19]. This project supports a prototype implementation of the ISMS specifications as part of the open source `Net-SNMP` package.

5.1 Presentation

The SNMP architecture defined in [15] and extended in [16] is designed to be modular in order to support future protocol extensions such as additional security models or additional transports. The architecture defines several subsystems and interfaces between these subsystems that should remain unchanged when subsystems are extended. The goal was to reduce side effects that can occur without such an architectural framework when the protocol is extended.

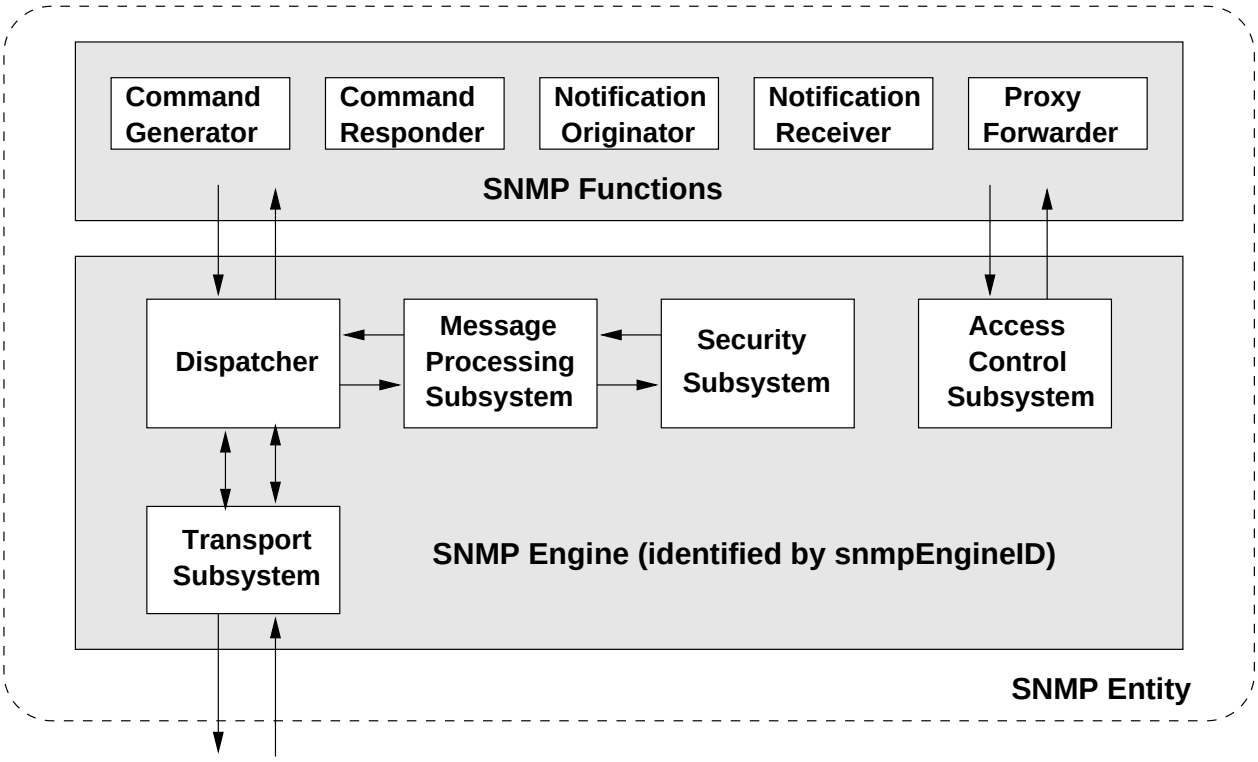

Figure 16: Structure of an SNMP entity according to the SNMPv3 architecture

According to the SNMP architecture defined in [15] and extended in [16], an SNMP engine consists of a message processing subsystem, a security subsystem, an access control subsystem, a transport subsystem and a single dispatcher (Fig. 16). Each subsystem can contain multiple concrete models that implement the services provided by that subsystem. The interfaces between subsystems are defined as Abstract Service Interfaces

(ASIs). The dispatcher is a special component controling the data flow from the underlying transports through the SNMP engine and up to the SNMP applications. The dispatcher is a singleton and a fixed part of the architecture.

Most existing SNMPv3 implementations support three message processing models for SNMPv1, SNMPv2c, and SNMPv3 and two security models, namely the User-based Security Model (USM) [20] (used by the SNMPv3 message processing model) and the Community-based Security Model (CSM) [21] (used by the SNMPv1 and SNMPv2c message processing models). There is only a single View-based Access Control Model (VACM) so far. In SNMPv1, SNMPv2c and SNMPv3/USM, security services (authentication, data integrity checking, encryption) are provided by the security subsystems, which is called from the message processing subsystem. This approach is called *message-based security* since all security processing happens on a per SNMP message basis. In particular, there is no notion of a security session and hence there is also no notion of session keys.

The transport subsystem [16], which has been recently defined by the ISMS working group of the IETF and was not part of the original SNMP architecture, permits multiple transport protocols to be ”plugged into” an SNMP engine. The transport protocols are represented by transport models and may be security-aware.

Particularly interesting are secure transport protocols such as SSH, TLS, or DTLS that are wrapped by corresponding transport models, namely the SSH Transport Model (SSHTM) [19] or the TLS Transport Model (TLSTM) [22]. These secure transport models have in common that they have a notion of a session and provide security services (authentication, data integrity checking, encryption) on a per session basis, so called *session-based security*. Furthermore, the secure transport protocols all have a notion of a session key, which is generated out of initial keying material and valid only for the duration of one session.

The SNMP architecture was not designed with session-based security in mind. As a consequence, the original ASIs between the subsystems do not pass all the necessary security information to all subsystems. In order to minimize the changes to the original architecture, a new Transport Security Model (TSM) [17] for the security subsystem was introduced. The TSM sits where traditionally message-based security services are provided and it interacts with session-based secure transports through a shared cache. The necessary translation of transport-specific security parameters (e.g., the name of the authenticated SSH user) and the SNMP-specific, model-independent parameters (e.g., `securityName` and `securityLevel`) happens within a secure transport model (e.g., the SSHTM or the TLSTM) while the transport security model (TSM) simply pulls this information out of the shared cache and puts the data into the corresponding ASIs. As a consequence of this approach, a single transport security model can be used with a variety of secure transport models.

5.2 Expected Impact

The software is meant to be a reference implementation for the specifications produced by the ISMS working group of the IETF. The implementation experience has already helped to gain some implementation feedback on different design choices. In addition, the imple-

mentation aims to serve as a research platform for prototyping and evaluating alternate secure transports.

In the longer run, we aim at the integration into the Net-SNMP package. However, this turns out to be not realistic before the ISMS working group of the IETF has finished the specifications as the source code changes are relatively large and it would not help ISMS if different versions of implementations of working group drafts are spread around. This is an important consideration since several vendors (e.g., Apple) rely on Net-SNMP code for the SNMP agent shipped with their operating system.

5.3 Progress Report

A prototype implementation of SNMP over SSH / TLS / DTLS has been developed at Jacobs University as an extension of the widely used open source Net-SNMP[1] SNMP implementation (version 5.3.0.1). For the SSH protocol, the open source libssh[2] C library was used. It contains all functions required for manipulating a client-side SSH connection and an experimental set of functions for manipulating a server-side SSH connection. For the TLS and DTLS protocol, the open source openssl[3] library was used.

The implementation itself consists of a new C module implementing the TSM (about 300 lines of code) and three additional C modules implementing the transport models SSHTM, TLSTM, and DTLSTM (each one roughly 900 lines of code). The Net-SNMP internal API for adding transports worked reasonably well and did not require any changes. The fact that Net-SNMP already supports stream transports was convenient. For password authentication, the SSHTM calls the Linux Pluggable Authentication Modules (PAM) [23] library to make it runtime configurable how passwords are checked.

Most of the development time was spend on optimizing the performance of the implementation since the overall latency initially was surprisingly high. In order to optimize the performance of the SSH transport domain, we investigated the influence of TCP's Nagle algorithm as well as the windowing mechanism of the SSH protocol. For the DTLS implementation, we had to investigate how to set the DTLS timers and patch a long known bug in openssl. Furthermore, we did run into some limitations of the Net-SNMP package we used.

5.3.1 TCP Nagle Interactions

During our initial measurements, we observed that the execution of a Get operation over the SSH transport domain required approximately $800ms$. This surprisingly large delay was introduced by TCP's Nagle algorithm, which essentially delays the sending of a TCP segment until either a segment has been filled or the previous segment has been acknowledged. We therefore disabled the Nagle algorithm by setting the TCP_NODELAY flag on the agent and on the manager side of the connection. This lead to a significant improvement in the performance of the SSH transport domain as the time required for the execution of a

[1] http://net-snmp.sourceforge.net/
[2] http://0xbadc0de.be/wiki/libssh:libssh
[3] http://www.openssl.org/

snmpget operation went down to $56.5ms$. We further modified the libssh library to disable the Nagle algorithm immediately after establishing the TCP connection between the agent and the manager and before any SSH exchanges take place. This further reduced the time required for a Get operation to $16.17ms$.

5.3.2 SSH Window Adjustments

The SSH windowing mechanism is used to specify how much data the remote party can send before it must wait for the window to be adjusted. In the OpenSSH implementation, window adjustment messages are only exchanged periodically. During our initial testing of our SSHTM implementatin, we noticed that each message exchanged between the agent and the manager was followed by a window adjustment message. These additional messages introduced significant bandwidth overhead as well as latency overhead for long sessions. As a result the TSM/SSH transport performed worse than the USM transports with respect to latency and bandwidth. In order to optimize the performance, we modified the libssh library to send window adjustment messages only when necessary. This improvement lead to better bandwidth and latency performance of the TSM/SSH transport compared to the USM transport.

5.3.3 DTLS Retransmission Timers

DTLS uses a retransmission timer to handle packet loss. The timers are managed internally by the openssl implementation. The API simply provides a function to set the timeout. While the API in general is easy to use, the question remains how this timer should be set. Note that SNMP engines usually have their own retransmission logic and hence it is crucial that these retransmission timers play well together. In the TCP based transports, this is less of an issue since TCP adapts the retransmission timers automatically and it is thus safe to turn the SNMP retransmission logic off.

The DTLS and SNMP datagram transports provides more flexibility concerning timers but with flexibility comes responsibility. With the current API, it is non-trivial to implement adaptive timers and hence implementations frequently fall back to fixed interval timers. This is true for both the Net-SNMP and the openssl implementation we used. In order to let DTLS take care of retransmissions, we have set the DTLS retransmission timer to a fraction of the SNMP retransmission timer but we have kept the SNMP retransmission timer since in principle just the DTLS datagrams carrying SNMP messages could have been dropped and DTLS does not provide a reliable data stream like TCP.

5.3.4 Net-SNMP Limitations

During the implementation, we noticed a number of limitations of the Net-SNMP package that are important also for the performance evaluation. The first limitation concerns the Net-SNMP retransmission timers, which by default can only be set in the resolution of seconds (with a default of one second). On many Internet links, waiting a second to start a

retransmission is rather ineffective. We changed the source code so that we can specify retransmission timers in milliseconds.

A second and more significant limitation concerns the handling of the `msgMaxSize` SNMPv3 message header field. This header field is of particular importance for the processing of `GetBulk` requests since it provides a hint for the maximum response message size that can be returned to the manager. Unfortunately, the `Net-SNMP` agent code does ignore this hint and thus it is difficult to control the response message sizes generated by an agent. It is even possible that the agent generates responses that do not fit into the buffers provided by the manager and are thus dropped.

The third issue is related to the handling of `contextEngineID`s. According to the SNMP specifications, an agent may provide access to multiple contexts and the `contextEngineID` and the `contextName` is used to select the appropriate context. A special context engine discovery procedure has been defined for use with the TSM [24]. During implementation, it turned out that the used `Net-SNMP` version actually treats any `contextEngineID` as referring to the local default context.

5.4 Conclusions

The ISMS reference implementation development funded by the EMANICS project has been important to ensure that the specifications developed in the ISMS working group of the IETF are implementable. The work also led to additional specifications, such as the SNMP EngineID discovery document [24], that are necessary to make the technology complete and implementations work. In addition, the implementation allows us to experiment with other secure transport models such as TLS and DTLS.

While the implementation is functional, there is additional work to be done as the working group proceeds. Since the `Net-SNMP` version we started from has been superseded by another major release, additional work is necessary to port our ISMS implementation to the current `Net-SNMP` code base. Since the ISMS implementation creates new dependencies on external libraries, work is needed to create proper cross platform configuration scripts. Finally, there is some work to be done on the integration with `Net-SNMP` command line and configuration file processing.

6 P2PFastSS

6.1 Introduction

Peer-to-peer (P2P) systems have evolved from unstructured systems like Gnutella to structured systems like Chord [25], CAN [26], Pastry [27], or Kademlia [28], which implement efficient distributed hash tables (DHTs). P2P systems can be used to overcome limitations present in centralized systems. Such limitations include low scalability, weak fault tolerance, and poor load balancing.

Basic operations in a DHT are get(key) and put(key, value). Data is stored in a DHT by applying a hash function to the key and calling the put method. A search is performed using the get method. The same hash function is applied to the query. The benefit of using a structured P2P system is that a key lookup typically requires O(log n) routing steps, where n is the number of peers in the system. However, DHTs have limited support for similarity search, because calculating a hash function on similar keys results in different hash values. DHT-based systems lack support for similarity search using the edit distance metric. Existing solutions are limited to n-gram based substring searches [29] and IR-type ranking based on the Jaccard coefficient [30].

P2PFastSS (fastss.csg.uzh.ch) is a content-based full-text fast similarity search algorithm that uses the edit distance metric and is applicable to the service discovery domain. The edit distance [31] is the minimum number of operations required to transform one string into another, using the operations: deletion, insertion, and replacement. The proposed algorithm can be used on top of any DHT, since it uses only the basic operations get and put.

P2PFastSS can work with any type of textual information. To test its functionality, abstracts of Wikipedia articles are used as documents. Both the title and the content of these articles did provide a base for similarity search. A number of articles were downloaded from Wikipedia in a structured XML file format and it was then provided to one of nodes in the P2PFastSS network.

6.2 P2PFastSS Architecture

A naive approach to similarity searching is to use broadcasting or flooding based P2P networks. Each peer in the network will receive the query and can locally execute a pattern matching algorithm. However, broadcasting or flooding typically do not scale well. This section describes the algorithms related to similarity search and related work that addresses scalability issues. Similarity search can be used with the edit distance, which is the minimum number of operations required to transform one string into another. An operation is either an insertion, a deletion, or a replacement. The edit distance is calculated using dynamic programming (DP) in $O(pq)$ time where p and q are the lengths of strings being compared. For example, the edit distance between house and mouse is 1 because the minimum operation to transform house into mouse is to replace h with m.

P2PFastSS finds text documents containing keywords similar to the ones included in the search query. Similarity in P2PFastSS is defined as the edit distance k between keywords.

P2PFastSS is built on top of a structured P2P network in order to achieve a lookup time logarithmic in the number of nodes. The algorithm is split into two phases. In the first phase the document is stored and indexed, while in the second phase the similarity search is performed.

6.2.1 Indexing and Storing (First Phase)

In the first phase documents in P2PFastSS are stored using the DHT operation put(key, value). The key of the document is the hash of the document title. The value of such an entry is the document itself (see Table 1). All words included in the document need to be indexed before a similarity search can be performed. Indexing is performed using the following steps.

Table 1: Document index excerpt of node 0x1235

Title (Key)	Document (Text)
Albedo (Doc ID: 0x123)	The albedo of an object ...
Achilles (Doc ID: 0x132)	In Greek mythology, Achilles ...
Paper (Doc ID: 0x238)	Paper is a commodity of thin ...
Testing (Doc ID: 0x321)	This test aims to ...

1. All words in the document are identified.
2. A deletion neighborhood is generated for all words in the document.
3. All neighbors are stored with a reference to the document IDs in which the word appears, using the operation put(key, value). As words may appear in many documents, a key can hold multiple values (a list of references).

In the following example, keys and values of index entries are shown. Table 1 shows the keys and values of a nodes document table. Table 2 shows keys and values of a nodes keyword index, and Table 3 neighborhood entries for test with k=1. The key is determined by the hash of the document title (Table 1) or keyword (Table 2). The value contains the text (Table 1), or the keyword and a reference that points to the document (Table 2). If a word appears in several documents, it is mapped to a list of locations.

Table 2: Keyword index excerpt of node 0x1235

Keyword (Key)	Resource and Location
object (hash 0x424)	object, Doc ID: 0x123...
pper (hash 0x927)	paper, Doc ID: 0x238...
	piper, Doc ID: 0x641...
wter (hash 0x149)	water, Doc ID: 0x583...
tes (hash 0x123)	test, Doc ID: 0x321...

During the indexing phase P2PFastSS generates a deletion neighborhood by using the method precalculate and will index all neighbors of word test (Table 2). The messages sent by the searching node in the process of indexing and storing a neighbor in a distributed environment are shown in Figure 17. Node 0x1 wants to index the deletion neighborhood of the word test (hash 0x563), which points to the document with ID 0x321. In messages 1 and 2, it finds the locations appropriate for placing the index entries for the first neighbor tes, which involves two routing queries. Then tes is stored redundantly in message 3. Messages 1 and 2 are sent sequentially, while messages 3 can be sent in parallel. This process continues until all neighbors and the keyword are stored.

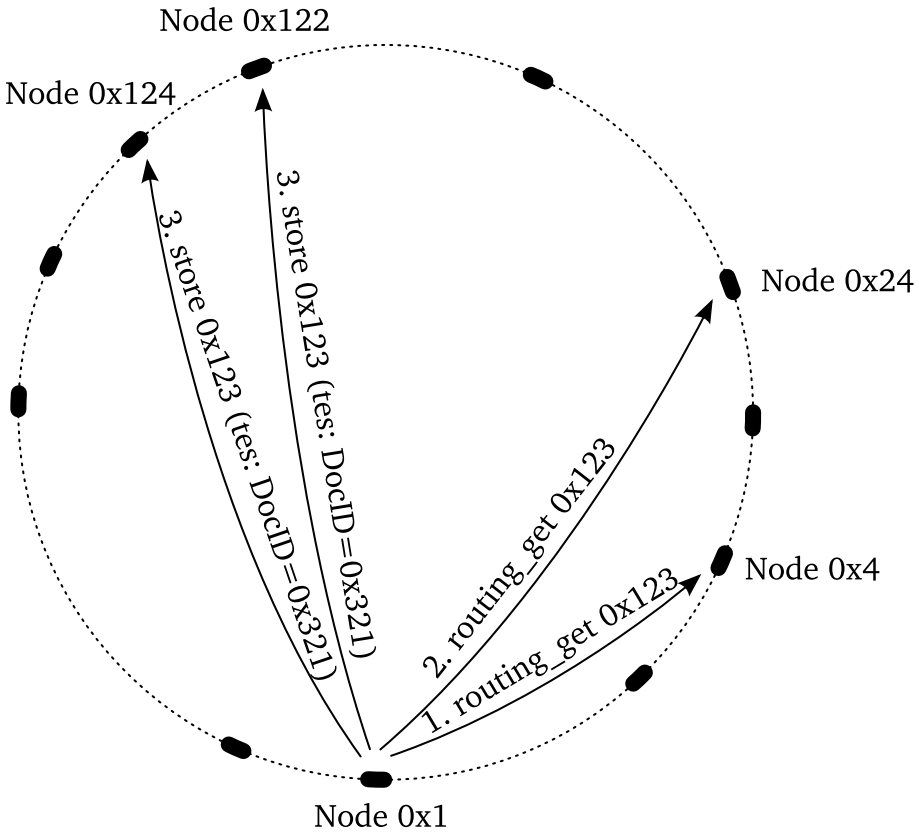

Figure 17: Message diagram of the indexing phase of tes (hash:0x123)

Table 3: Keyword index excerpt of node 0x1235

Keyword (Key)	Resource and Location
object (hash 0x424)	object, Doc ID: 0x123...
pper (hash 0x927)	paper, Doc ID: 0x238...
	piper, Doc ID: 0x641...
wter (hash 0x149)	water, Doc ID: 0x583...
tes (hash 0x123)	test, Doc ID: 0x321...

6.2.2 Searching (Second Phase)

The similarity search is performed in the second phase using the following steps.

1. A node generates a deletion neighborhood from the search keyword.
2. Every neighbor is searched for using get(key). The result contains a document ID and the keyword. Before the document is retrieved using get(key), potential matches are confirmed using DP (see Section II) to check if they do not exceed the given edit distance.
3. The document is retrieved with get(key) using the matching document IDs found in step 2.

The message diagram for the search in a distributed environment is shown in Figure 18. Node 0x1 searches for the word tesa. Before searching, P2PFastSS generates a deletion neighborhood from tesa: (tes,tea,tsa,esa) and starts by searching for the first neighbor tes. The key for tes is 0x123. First, the node address is found via messages 1 and 2. In message 3, the node containing tes is looked up. Messages 1, 2, and 3 are sent sequentially. Finally, node 0x122 returns the Doc ID, which can then be used to retrieve the document containing the term tes which is similar to tesa (edit distance 1)

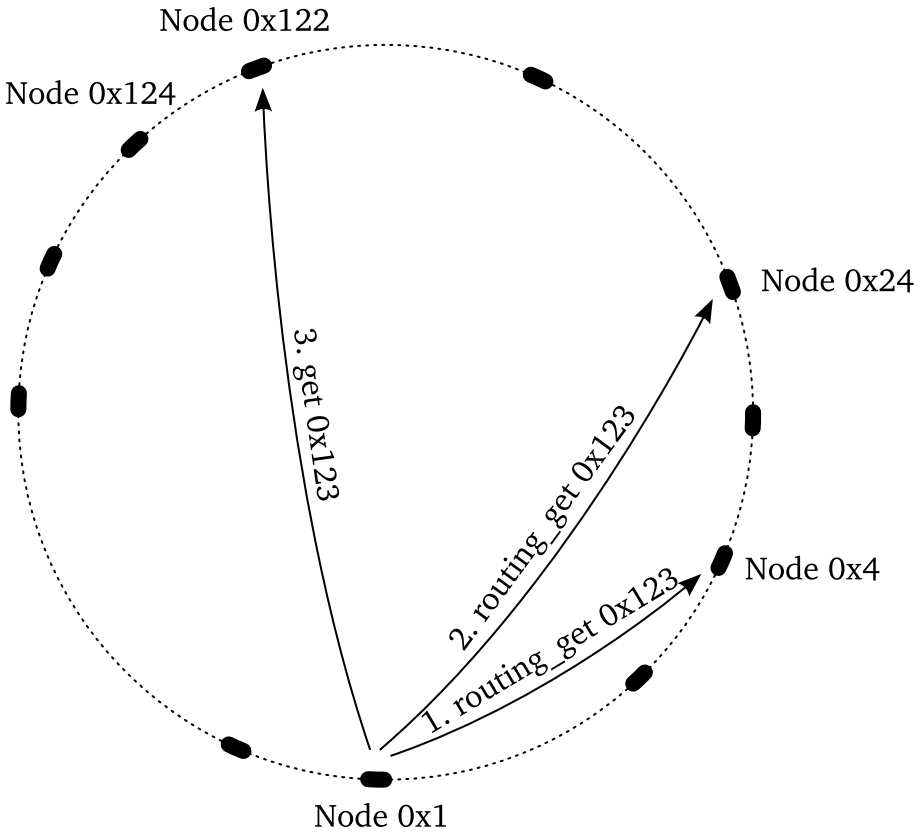

Figure 18: Message diagram of the searching phase of tes (hash:0x123)

6.3 P2PFastSS Changes and Extensions

P2PFastSS does not use the data itself to perform similarity search — it needs to build an index of the data before it is able to search. Besides searching, any node has storing and indexing capability. However, creation of the index was not distributed; only one node was responsible for it, while others were idle. Therefore, it could take a long time to index large datasets. This behaviour has been changed, allowing any node to distribute indexing effort randomly to other nodes. With distributed indexing, an initiating node is still needed, since a specific node must coordinate and distribute articles for other peers to index.

Distributed index works as follows: 1. an initiating node extracts articles from the XML file and uses the DHT put operation to write them into the DHT, effectively sending it to a random peer in the network; 2. any nodes that receive a full article must create an index with all words in the article, plus their deletion neighborhood for the constant edit distance; 3. those peers, after building the index, use the put operation to store every generated deletion neighborhood into the DHT.

Once the deletion neighborhood was computed, a key containing the result of a hash function applied on the articles title was computed and the article was stored into the DHT by using put(key, article). A normal DHT operation, put routes and stores the article in a node with node ID closest to the key. A slightly different procedure was used to store the indexed data. For each deletion neighbor of a given word, a special object, called KeyAndKeyWord, was created. This object contained the original word, from which the deletion neighbor was generated, and the title of the article from which the word belongs. The reason behind this approach is that different words can generate equal deletion neighbors; hence, a way to connect the index data with its original target data is needed. Once the KeyAndKeyWord object was created and initialized, it was put into the DHT. There was no logical separation in code for article storing in DHT and their indexing: it was all done by a single method storeArticles(). The path to the XML file could only be passed as argument before running the program: once the XML file was processed and the articles indexed, no other articles could be added while the program was running. The software was extended with a web interface which allowed selecting of the XML file to process, manual insertion of articles into the system, submitting of search queries and displaying the search results.

Each node had separated data structures for managing the storing procedure of articles and of the index data. Both structures used implemented the Map interface. For the articles, two ConcurrentHashMaps were used, together with following Genenrics as (key, value) pair: (BigInteger, Map¡String, Object¿). The reason behind this “map-in-map” approach was to allow the user to specify an additional domain as a String, allowing storing more objects for a single key.

A first action was to logically separate the various tasks, by using different Java methods for each of them. So the storeArticles() method was replaced by removing the code that was not involved with article storing. A new method for article indexing purpose, indexArticle(), was created. The method parameters are the article to index, and an integer to indicate the edit distance desired for the indexing. This method creates KeyAndKeyWord objects with generated deletion neighbors and passes them to the DHT for storing. To avoid blocking a node while it is indexing articles, a dedicated thread, called IndexRunner, is added to the P2PFastSS node. It runs in parallel with the node and takes care of indexing. When there are no articles to index it can be but in sleep mode. Different solutions

were considered regarding how could a node could differentiate what type of objects it stores in his map. Initially, message payload analysis and reengineering of the class managing the article's map (StoreMultiMapMemory), to allow callbacks on the nodes main class, was considered, but those solutions would have been complex and inelegant. A much more elegant solution is to slightly modify StoreMultiMapMemory, to allow listeners to be added to it. A last update to this class code was to notify all its listeners whenever an article was going to be stored on the map. Finally, node's constructor was modified, so that an anonymous listener is created and added to the node's StoreMultiMapMemory. This way, whenever an article is stored, eventual listeners are notified. They then send the article to the IndexRunner for indexing and putting the index data in the DHT.

6.3.1 Disk-based Storage

The former implementation of P2PFastSS used peer's volatile memory to store the data. While this allowed fast access to the stored information, it had two main drawbacks:

1. Given the nature of RAM, once the P2PFastSS application running on a node was closed (intentionally or not), all article and index data stored in that node were lost. If the whole P2PFastSS network was shut down, all articles would have to be processed again to return to the previous state.
2. Index data size tends to be multiple times the size of the text. In [32] it is stated that a 388 KB dictionary results in an approximate 100 MB index for an edit distance of 2, which is more than 250 times larger. This situation causes the P2PFastSS application to run out of system memory.

By implementing a disk-based storage, these drawbacks could be countered: nodes could rejoin the P2P system after failure, and the system could resume after a shutdown, without need to re-index all the data.

Before implementing the disk-based storage for P2PFastSS, an important decision had to be made. It concerned the choice of storage system to be used: the file system (FS) or a database management system (DBMS). Table 4 shows some key benefits and drawbacks considered for both systems. Benefits and drawbacks of File Systems and Database Management Systems Benefits Drawbacks FS Simplest approach: Java Serialization could be used. Special file format could be used. Can be compressed. (DB also) Java Serialization is slow and sensitive to versioning; size of serialized data usually larger then objects size in memory. If special format used: parsing of file is necessary. Read / write access must be managed and synchronized, as both can happen concurrently DBMS Standardized Language (SQL). Open source DBMS software available. Complex queries possible DBMS responsible for concurrent read/write access Flexibility in accessing database table fields. DBMS responsible for data integrity CPU overhead from DBMS threads Memory overhead from DBMS threads

It was decided to use a relational database manager. HSQLDB was used for the following reasons: It is written in Java and offers JDBC APIs, it is available under BSD license, library size is small (around 600 KB for the standard edition), no need to start the server in separate JVM (can run in In-Process mode).

Table 4: Benefits and drawbacks of File Systems and Database Management Systems

	Benefits	Drawbacks
FS	- Simplest approach: Java Serialization could be used. - Special file format could be used.	- Java Serialization is slow and sensitive to versioning; size of serialized data usually larger then objects size in memory. - If special format used: parsing of file is necessary. - Read / write access must be managed and synchronized, as both can happen concurrently
DBMS	- DBMS responsible for data integrity. - Open source DBMS software available - Complex queries possible - DBMS responsible for concurrent read/write access. - Flexibility in accessing database table fields. - Standardized Language (SQL)	- CPU overhead from DBMS threads - Memory overhead from DBMS threads

6.3.2 Keyword Relevance

To test the correct running of the P2PFastSS system and to allow a simple view of the search results, a simple web interface is used; it allows the user to select XML files for indexing, to perform similarity search and to display the results. However, the results weren't sorted: they were displayed in the order the various nodes answered to the search queries. This caused situations in which results with an exact match (with edit distance 0) were displayed after some results with larger edit distance. In order to sort results by relevance, a numerical value representing the relevance of a keyword in an article has to be calculated.

A "relevance function" was added, which assigns, at indexing time, a numerical value to each deletion neighbor calculated. The factors that influence the relevance score are edit distance, whether the keyword appeared in the articlestitle. The relevance score is saved with each generated deletion neighbor and is used as the base for sorting the search results before showing them on the web interface. The list below enumerates some important properties that have impact on a keyword's relevance: Edit distance (the lower, the greater relevance), number of occurrences of the keyword in the article, length of the keyword (in characters), total words contained in an article, occurrence in articles title (if the keyword appears in the title the score should be increased).

Since the application must store, with each deletion neighbor of an indexed word, its relevance score in an article, the KeyAndKeyWord class is modified to carry the additional relevance variable.

6.3.3 Paging of Search Results

In the previous version, all the results were displayed in a single page. Whenever a search query produced many search results, this page would result higher than the user's screen, so he or she would have to scroll the page to check all the results. If the results had to be verified one at a time, then the user had to visually keep track of the last verified result, which could be tedious and frustrating for the user. Another issue was the results page loading time, where a page displaying many results would take more time to be visualized. For this reason a paging feature was implemented, so that the search results would have been separated into multiple sets.

Only one set of results would be displayed on the results page at a given time, and the user can decide to navigate back and forth trough the sets. The sets would have to be sorted by relevance so that those displayed first would contain the results with higher relevance score then those displayed later. The user browses the search results by selecting the previous or next link. Navigation trough the results should not imply new search queries. The first page of search results does not show the previous link, while the last page does not show the next link. A default number of results per page is predefined and if the result set is smaller or equal to it, both links are not displayed.

To obtain the sorting of search results, they need to have an ordering relation. The most natural ordering relation is their keyword relevance score. By the time this particular implementation has begun, the keyword relevance function was already implemented and available. But the KeyAndKeyWord-objects still could not be compared between each other. To make them comparable, it was decided to modify KeyAndKeyWord class so that it would implement the Comparable interface. Consequently, the method compareTo() is implemented, making KeyAndKeyWord-objects comparable by relevance. A new collection type is used for containing the results of search queries to avoid manual sorting. The Java class TreeSet is chosen for this task because it automatically sorts the elements it contains by using their compareTo() method, and updates the order if needed when elements are added or removed. Finally, the code for submitting queries was logically separated from the code responsible for displaying the results. The results of a search query are temporarily saved in a TreeSet object and this object is then iterated to obtain the results for the displayed page. Iteration of the TreeSet does not produce new search queries or additional network traffic.

6.3.4 Preview of Search Results

The visual output for the search results displayed in the web interface consisted only of the articles title and a numerical hash value of it. If the title was not informative enough for the user, he had to open several articles, until the searched article was found. If the articles were not on the node that user was using, then the node had to fetch many of them from the DHT, thus generating network traffic. It is assumed that, if the user was given more information about the search results, he or she would identify the searched article (or other type of resource) faster and it would generate less network traffic.

To present better contextual information to the users, each result must include a segment containing the keyword in the article. It was decided to extend the software so that when

a keyword is indexed, its position in the article is also to identified and a short segment of text containing the keyword, defined as the preview string, is saved together with the index results.

The KeyAndKeyWord class is furthermore extended to allow saving of the preview string inside of it. At index time, for each generated deletion neighbor, a portion of the articles text, containing the original keyword and a given number of surrounding words from the left and from the right side, is saved as the preview text. To allow the preview string to be saved on disk, the database table for index data is extended.

The web interface is modified to show the preview text of the article, in addition to the articles title. Before actual displaying, the search results are processed, the keyword is identified in the preview string and HTML bold tags are applied to it, as Figure 19 shows, to make the keyword even more visible to the user.

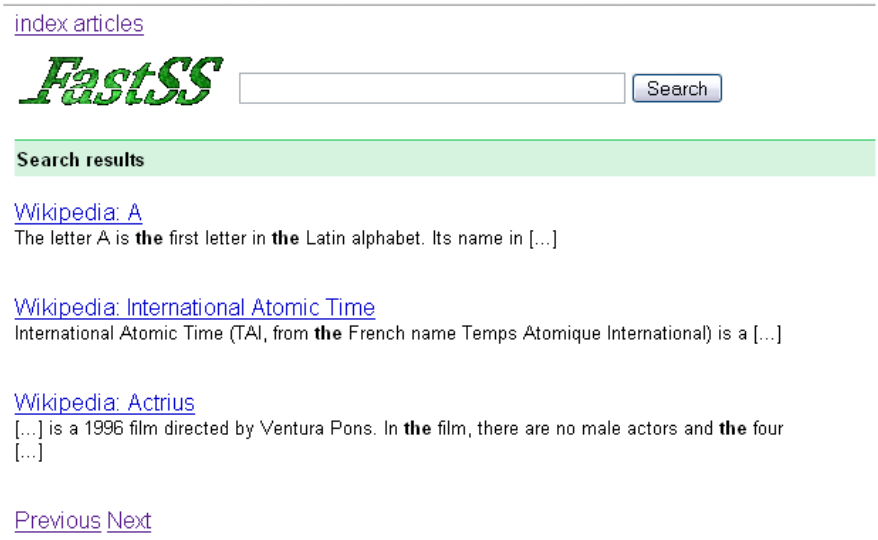

Figure 19: P2PFastSS screenshot

6.4 Impact Evaluation

P2PFastSS and TomP2P are in an early stage and the first version has been released to the project site (http://tomp2p.csg.uzh.ch/). The project page is expected to be a source of knowledge for similarity searches in P2P networks. In future, mailing lists and forum will be added for public discussions about the development of similarity searches in P2P networks. Current developers of P2PFastSS are: Thomas Bocek (UZH, lead), Fabio Hecht (UZH), and Dalibor Peric (UZH).

7 NDPMon packaging and distribution

7.1 Presentation

Please refer to section 4.1 for a presentation of the NDPMon framework.

7.2 Expected Impact

Ensuring standard organization and availability for most common operating systems will increase the number of potential users and extend the community around the tool, bringing more feedback and visibility. A binary distribution, especially if the package is integrated in the official repositories, will also increase these two points. Finally, we expect to ensure the durability of the tool thanks to these operations, by developing the developers community, which will emerge from these tasks.

7.3 Progress Report

In this section, we will present the progresses and new features brought to NDPMon since the beginning of the EMANICS support campaign regarding organization and distributions.

File System Hierarchy Standards

The first step towards packaging and distribution is to meet the standards especially the File System Hierarchy Standards (FHS - `http://www.pathname.com/fhs/`), in order to place all the files in their right place, and ensure the compatibility. The default paths used by NDPMon are compliant with the FHS since version 1.3 from the 10 September 2007.

By default, the files are installed in:

- /usr/local/ndpmon: sources, plugins and scripts
- /usr/local/etc/ndpmon: configuration file
- /usr/local/share/man/man8: manpage
- /usr/local/sbin: executable
- /var/local/ndpmon: variable data (discovery history, neighbor cache, alerts)
- /etc/init.d: startup script

All these paths can be changed via the configure script before building the tool.

Binary Packages

In order to ease the distribution on the tool, we enabled binary distribution for Debian and FreeBSD.

NDPMon is available as a FreeBSD port (`http://portsmon.freebsd.org/portoverview.py?category=net-mgmt&portname=ndpmon` and `http://www.freebsd.org/cgi/cvsweb.cgi/ports/net-mgmt/ndpmon/`). Janos Mohacsi from HUNGANET currently maintains this port.

Debian packages are also available for AMD64 and i386 architectures via the tools main download site (`https://sourceforge.net/project/showfiles.php?group_id=178666&package_id=247867`). These packages are maintained within the team.

SourceForge Project

NDPMon is distributed under the LGPL licence as a SourceForge project. The project itself is available at the URL `https://sourceforge.net/projects/ndpmon/`. NDPMon also has an official WEB site at the URL `http://ndpmon.sf.net/`. Having a SourceForge project ensures the tools visibility, and eases the creation and management of the developers community. At the moment, the NDPMon community is composed of 4 members:

- Frederic Beck (INRIA/LORIA, France) - Maintainer
- Thibault Cholez (INRIA/LORIA, France)
- Pierre Chifflier (INL, France)
- Janos Mohacsi (NIIF/HUNGARNET, HUNGARY) - FreeBSD Port Maintainer

The community also welcomed the contributions of:

- Steffen Strueber (TU Clausthal, Germany): new alerts systems with pipes, corrected warnings
- Maik Frchtenicht (TU Clausthal, Germany): ndpmon2html

In the SourceForge project page, all the information about the tool is available, such as statistics on the projects activity, project details, screenshots SourceForge is also hosting the sources repository and code version via a Subversion repository, which can be browsed via HTTP. Besides this source access, the tools releases are available for downloading on both the WEB site and SourceForge via several packages:

- ndpmon: the sources of the tool
- ndpmon-extras: ndpmon2html and statistics
- ndpmon-web-interface: the WEB interface

- ndpmon-packages: binary distribution

For support, two mailing lists are available:

- ndpmon-users: general discussions about the tool and help to the users
- ndpmon-devel: discussions between developers and Subversion reports

NDPMon also has 2 forums and a bug tracker. All documentation and how-to are available on the official WEB site.

8 OpenDiameter Packaging and Distribution

University of Zurich (UniZH) is actively involved in providing OpenDiameter binary packages for Linux distributions Ubuntu [33] and Debian [34].

8.1 OpenDiameter

OpenDiameter [35] is an open-source implementation of the Diameter protocol and several applications based on top of it. The implemenation of the protocol is based on RFC3588 [36] designed by the IETF's AAA Working Group. The source code of the OpenDiameter software is available under the combination of Lesser GNU Pulic License [37] and GNU Public License [38]. The base protocol implementation is available as a C++ library and has currently support for Linux, BSD and Windows systems. It relies on ACE [39] to provide system level abstraction for all supported systems.

The libraries that OpenDiameter provides are:

- *libdiamparser*: Diameter message parser library (with XML dictionary support), with capability of user-defined AVP type parsers. As all configuration options are kept in XML files this library offers support for reading those files.
- *libdiameter*: Diameter core engine library with base accounting support. Based on this library further Diameter applications can be built. The library includes a light authentication and authorization application as well as an accounting application.
- *libdiametereap*: The Diameter EAP (Extensible Authentication Protocol) Application library may be used to build authentication applications based on EAP.
- *libeap*: The EAP library implements different EAP payload types and their handling.
- *libdiameternasreq*: The Diameter NASREQ Application library implements the NASREQ Application. It provides AAA services for dial-in PPP users and is the next generation replacement for the RADIUS protocol.
- *libdiametermip4*: The libdiametermip4 library provides a C++ API to Diameter MIP v4 (Mobile IP v4 Protocol) Application. The library implements the specification defined in draft-ietf-aaa-diameter-mobileip-20.txt.
- *libpana*: It implements the Protocol for carrying Authentication for Network Access (PANA). It is offered in IPv4 and IPv6 support for Linux, FreeBSD and Windows.

8.2 Expected Impact

Installation of OpenDiameter libraries from source may be a lengthy process due to its dependencies on different libraries. Having the OpenDiameter libraries as binary packages helps developers by offering an easy and fast way of installation. The interest for the OpenDiameter binary packages was observed on the CSG@IFI webpage hosting those

packages. The number of page hits is constantly increasing, currently having about 200 monthly hits. A set of EU-funded projects (such as Daidalos [40], EC-GIN [41], SmoothIT [42]) are using OpenDiameter for their research and are expected to make use of the OpenDiameter packages. Other EU-funded projects already used these packages (such as Akogrimo).

8.3 Progress Report

The OpenDiameter binary packages are already available on the CSG@IFI webpage [43]. The packages are based on the source code available from the OpenDiameter developers. Whenever a new version of OpenDiameter is released it is packaged for the most recent Debian and Ubuntu distributions.

OpenDiameter libraries are organized in three packages:

- *libdiameter1.0.7*: Includes all the OpenDiameter shared libraries. Those shall install under */usr/lib*.
- *libdiameter-dev*: Includes OpenDiameter header files used for development of applications on top of OpenDiameter. The files shall install under */usr/include/opendiameter*.
- *opendiameter*: Includes the *aaa/nas/pac* example binaries and respective configuration files. These examples are the one that OpenDiameter developers included in the source package for testing OpenDiameter.

Currently UniZH offers OpenDiameter packages for several Linux distributions and several OpenDiameter versions. The available packages by February 20th, 2008 are:

- Ubuntu Gutsy - OpenDiameter 1.0.7-h and 1.0.7-i
- Debian Etch - OpenDiameter 1.0.7-h and 1.0.7-i
- Ubuntu Dapper - OpenDiameter 1.0.7-h
- Ubuntu Breezy - OpenDiameter 1.0.7-h
- Ubuntu Hoary - OpenDiameter 1.0.7-h
- Debian Sarge - OpenDiameter 1.0.7-h

8.4 Debianizing Open Diameter

In order to ”debianize” the OpenDiameter software, the following steps had to be undertaken:

- First, the required Debian tools had to be installed for creating the packages, using the following command: `apt-get install dpkg-dev dh-make fakeroot`.

```
Source: opendiameter
Priority: optional
Maintainer: David Hausheer <hausheer@ifi.unizh.ch>
Build-Depends: debhelper (>= 4.0.0), autotools-dev, libssl-dev,
  libxerces26-dev, libace-dev, libboost-dev
Standards-Version: 3.6.1
Section: libs

Package: opendiameter
Section: net
Architecture: any
Depends: ${shlibs:Depends}
Description: Diameter binaries and documentation

Package: libdiameter-dev
...

Package: libdiameter1.0.7
...
```

Figure 20: `control` file

- Afterwards, a Debian package template was generated using the `dh_make` command, following the instructions given in the Debian Package Maintainers' Guide [44]. This resulted in a folder called `debian` within the OpenDiameter folder.
- As a next step, the corresponding configuration files within the `debian` folder had to be edited, namely the `control` file and the `rules` file. The `control` file is depicted in Figure 20, while Figure 21 shows the commands to configure the package in the `rules` file.
- Now the package installation files had to be created for the 3 different packages. Those are depicted in Figure 22.
- The OpenDiameter copyright notice was included in the `copyright` file and a `changelog` file was created, which has to be updated for every new version. A changelog entry example is shown in Figure 23.
- Finally, the package was compiled using the command `dpkg-buildpackage -rfakeroot`. Any rebuild can be done with the command `fakeroot debian/rules binary`.

8.5 Debian Repository for Open Diameter Packages

In order to create a Debian repository to download the Open Diameter packages, the following steps were necessary, as described in [45]:

```
config.status: configure
        dh_testdir
        CFLAGS="$(CFLAGS)" \
        XERCESCROOT=/usr/include/xercesc \
        ACE_ROOT=/usr/include/ace \
        BOOST_ROOT=/usr/include/boost \
        ./configure \
        --host=$(DEB_HOST_GNU_TYPE) \
        --build=$(DEB_BUILD_GNU_TYPE) \
        --prefix=/usr \
        --enable-shared \
        --enable-static
```

Figure 21: Commands to configure the package in the `rules` file

```
libdiameter-dev.dirs:
 usr/lib
 usr/include
 usr/share/doc/libdiameter-dev

libdiameter-dev.install:
 usr/include
 usr/lib/lib*.a
 usr/lib/lib*.so
 usr/lib/*.la

libdiameter1.0.7.dirs:
 usr/lib
 usr/share/doc/libdiameter1.0.7

libdiameter1.0.7.install:
 usr/lib/lib*.so.*

opendiameter.dirs:
 etc/opendiameter
 usr/share/doc/opendiameter

opendiameter.install:
 usr/bin
 usr/etc/opendiameter    etc/opendiameter
 usr/share/opendiameter/doc      usr/share/doc/opendiameter
```

Figure 22: Package installation files

```
opendiameter (1.0.7-f-1) unstable; urgency=low

  * Initial Release.

 -- David Hausheer <hausheer@ifi.unizh.ch>  Thu,  22 Feb 2008 13:59:03 +0100
```

Figure 23: Changelog entry example

```
dists
+-etch
  +-main
    |-binary-i386
    +-source
```

Figure 24: Folder tree layout

- A folder tree layout had to be created as shown in Figure 24.
- An archive config file `apt-ftparchive.conf` had to be generated. This is depicted in Figure 25.
- For every distribution a release config file `apt-<distr>-release.conf` was created. This is shown for Debian etch in Figure 26.
- To generate the repository, the following command was executed: `apt-ftparchive generate a` And for every distribution as shown for etch: `apt-ftparchive -c apt-etch-release.conf release /home/csg/debian-csg/dists/etch > /home/csg/debian-csg/dists/etch/Release`
- Finally, to add install the OpenDiameter packages, the following line has to be added to `/etc/apt/sources.list`: `deb file:/home/csg/debian-csg etch main`

```
Dir {
  ArchiveDir "/home/csg/debian-csg";
};

BinDirectory "dists/etch/main/binary-i386" {
  Packages "dists/etch/main/binary-i386/Packages";
  Contents "dists/etch/Contents-i386";
  SrcPackages "dists/etch/main/source/Sources";
};

Tree "dists/etch" {
  Sections "main";
  Architectures "i386 source";
};
```

Figure 25: Folder tree layout

```
APT::FTPArchive::Release::Origin "debian-csg";
APT::FTPArchive::Release::Label "debian-csg";
APT::FTPArchive::Release::Suite "etch";
APT::FTPArchive::Release::Codename "etch";
APT::FTPArchive::Release::Architectures "i386 source";
APT::FTPArchive::Release::Components "main";
APT::FTPArchive::Release::Description "Etch debian packages for csg";
```

Figure 26: Release config file `apt-etch-release.conf`

9 An online catalogue of available software for network management

This chapter describes only the work done after changes made into the Emanics Repository & Inventory application to use together with the Simpleweb.org repository (hosted at UT), what solved the database replication and coherence maintenance problems. All the work done before was described in the D6.1 and D6.2 - Emanics Deliverables.

For the Emanics phase II the following objectives had been pointed out within the software database extension joint UT and PSNC proposal:

- Any needed work related to current maintenance of the repository database and GUI web-application;
- Research and development works extending the existing repository's functionality, divided into the following two tasks:
  - Base extension for annotation support;
  - Classification of existing registered packages.

According to the aforementioned objectives during the Emanics phase II PSNC has performed some necessary maintenance works concerning Emanics repository web-application and database. PSNC made a few corrections in the application, which have been causing small problems with repository visualization in some of the web-browsers using another font set. All these corrections were thoroughly tested by PSNC before putting into practice in the official project web site.

In the last quarter of the year 2007 PSNC has initiated the discussion and cooperation with all project partners to choose the best way for repository extension. During this discussion and opinions exchange we proposed also our own technical and organisational ideas and schemas how to attain the intended objectives. As a result a proposal of about 10 the most important software packages was made which are considered as worth of an additional annotation in the existing repository. PSNC proposed the initial version of network management packages list, which was extended by project partners and finally accepted. At the end of this process PSNC proposed the following list of the best software packages designated for an additional annotation in the repository:

- cacti
- rrdtool
- net-snmp
- ntop
- wireshark
- cfengine

- nfsen
- nagios
- nmap
- flowtools
- fprobe

In addition to the above proposal we also suggested including several tools developed or ported by EMANICS partners especially in the scope of WP6 T6.3 activity (e.g. Ponder2, SCLI). As a result of our proposal the database hosted at UT was properly updated and the new list of the best and commonly used management software packages selected from the Emanics repository and designated for an additional users annotation is now available at the project's official web site: http://emanics.org/component/option,com_dbquery/Itemid,93/

The example view of this web page is presented below on Figure 27.

At the end of year 2007 PSNC designed, developed and tested the first version of automatic software repository classification. This new functionality of the Emanics repository performs automatic software assignment to the standard task's categories. This allows users to look through the repository not only using the search engine but also as an online catalogue of software divided into different functionality's categories. PSNC proposed to divide this software packages into the five categories in pursuance of the standard described in the TMN/Network Management Model proposed by the ITU-T and ISO organizations. These standard categories are known as ,,FCAPS" which stands for Fault, Configuration, Accounting, Performance and Security. The first programming effects of such new functionality were available to local tests at the beginning of year 2008. Then, after these tests and needed corrections PSNC implemented this functionality also in the operational and official Emanics web site in February 2008. It is publicly available at: http://emanics.org/content/view/122/146/ and the example view of this web page is shown on the Figure 28.

Below, an example of one out of the five categories is also presented to show the potential of the new Emanics repository functionality. The part of "Performance management" group is shown at Figure 29.

PSNC has also updated the "Emanics Inventory" table to reflect properly the latest database changes coherent with our proposal of the best software list, described earlier. The current view of this inventory is shown below on the Figure 30.

All the new and old functionalities of the software repository, introduced already in its operational version, were fully integrated by PSNC into the official EMANICS web site and are available publicly at the "Software" position of the main menu or via the direct link: http://emanics.org/component/option,com_dbquery/Itemid,93/

Figure 27: The new list of the best management software packages selected from the Emanics repository and designated for an additional annotation.

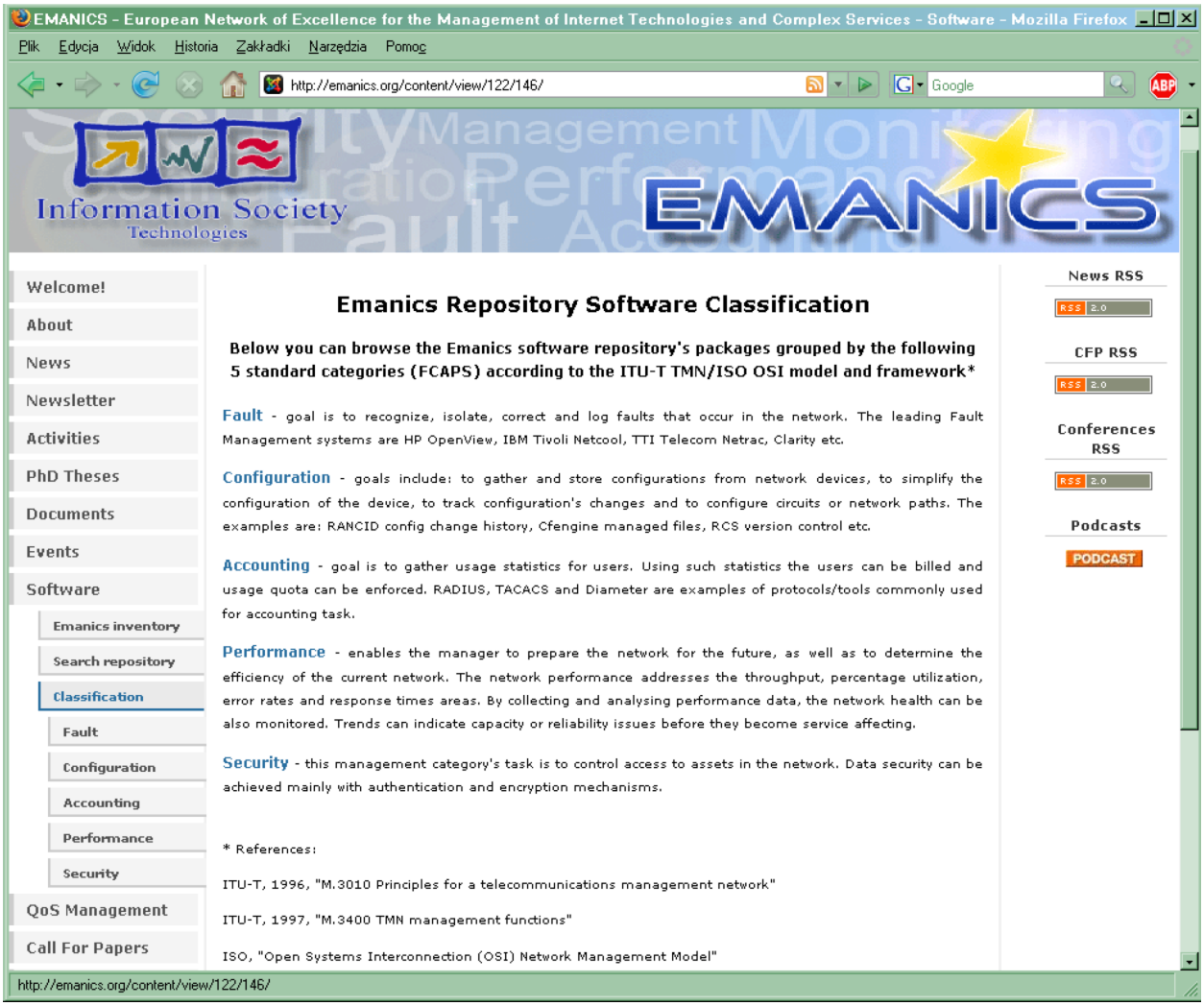

Figure 28: The main web page of the Emanics Repository Software Classification.

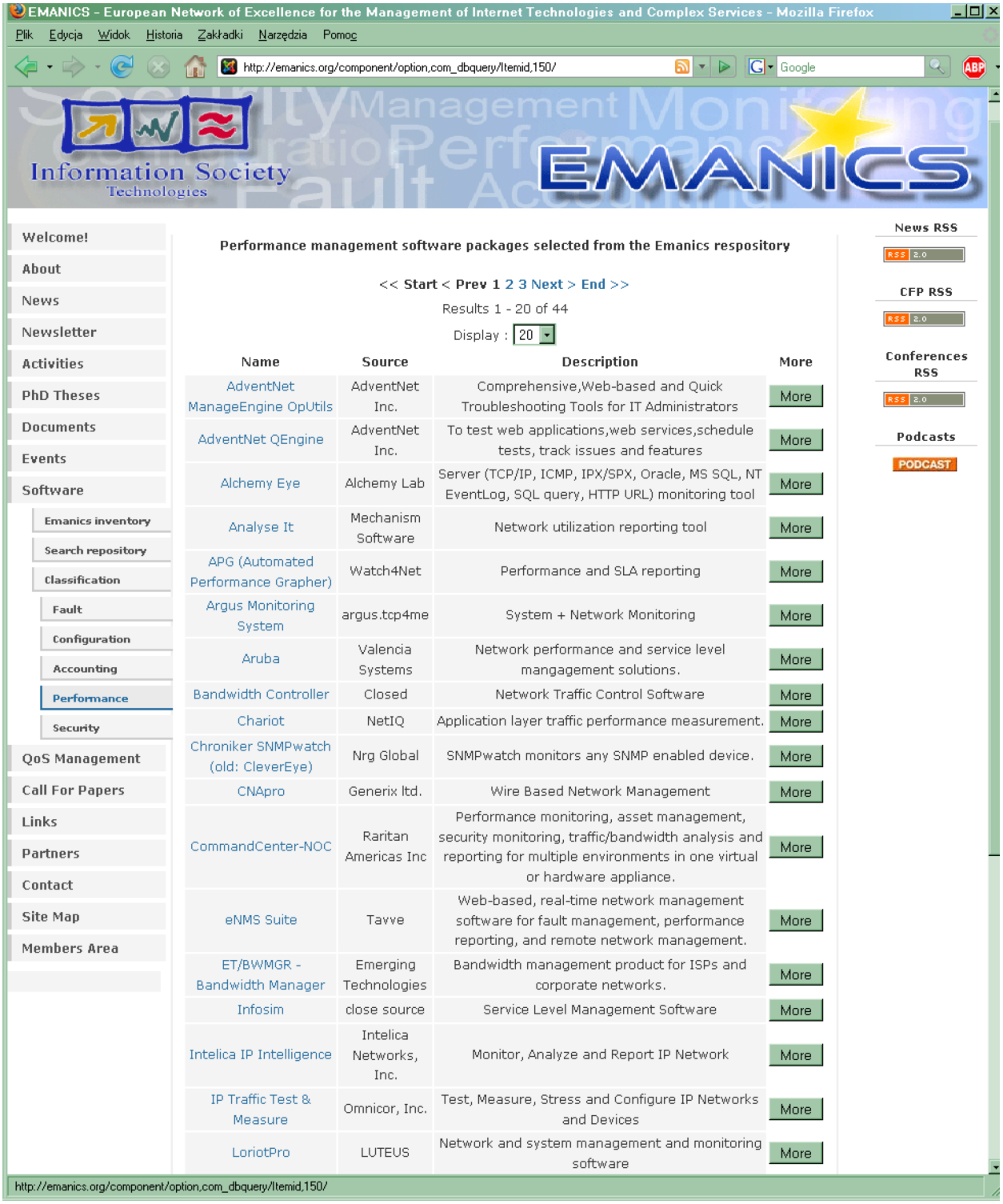

Figure 29: An example view of tools classification for the “Performance management” category.

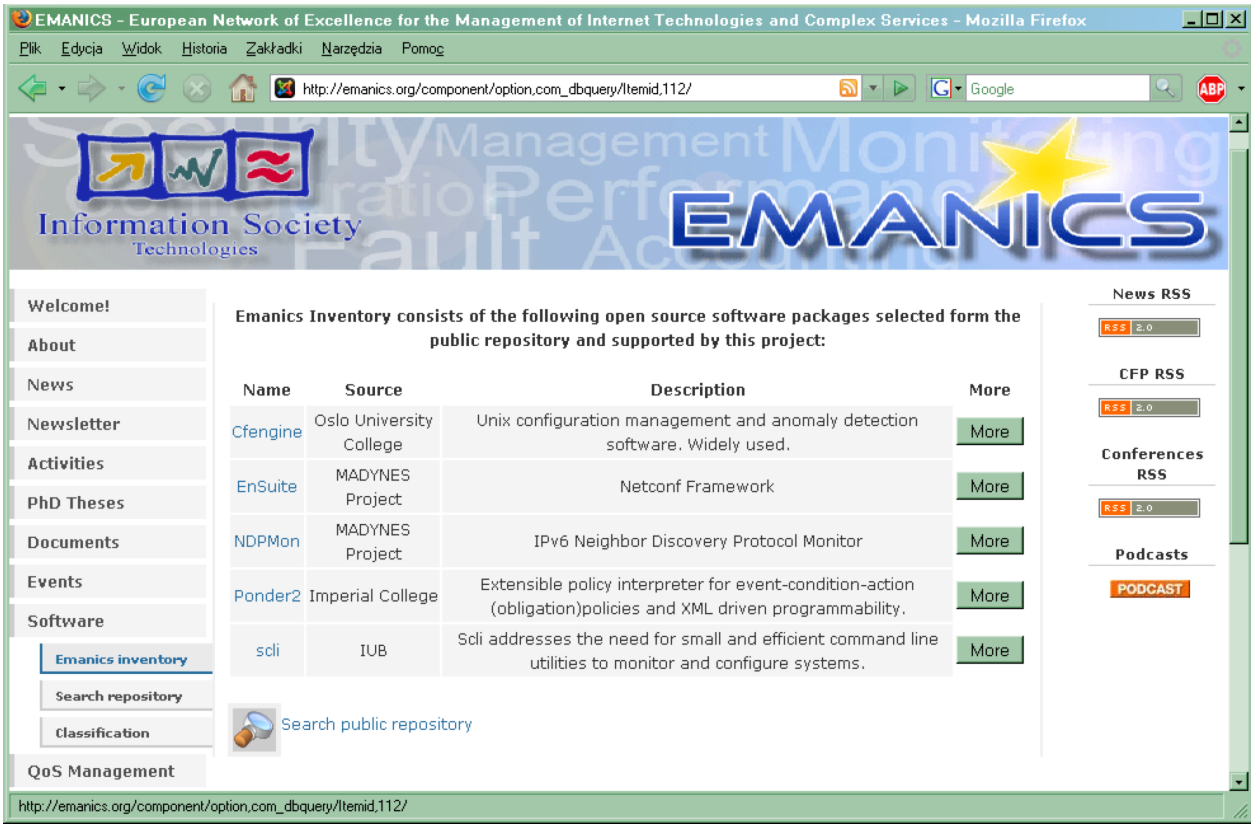

Figure 30: The updated Emanics Inventory web page.

10 Conclusions

Work-package 6 did, in phase 2 of the network of excellence, extend its activity with a third initiative dedicated to existing Open Source software packaging and tutoring. Dedicated software development/enhancement and inventory activities have been followed as well. As a result of the first call, seven activities among which two in packaging and one in inventory extension have been supported by work-package 6. 6 of these seven activities have been completed and related software packages, web-pages, distributions are all public. The last activity has just started.

A new call was issued this month. The supported initiatives will be reported in the next release of this deliverable which will be Deliverable D6.4 to be provided in December 2008.

11 Abbreviations

12 Acknowledgement

This deliverable was made possible due to the large and open help of the WP6 Partners of the EMANICS NoE. Many thanks to all of them.

References

[1] F. Strauß and J. Schönwälder. SMIng - Next Generation Structure of Management Information. RFC 3780, TU Braunschweig, IU Bremen, May 2004.

[2] M. Rose and K. McCloghrie. Concise MIB Definitions. RFC 1212, Performance Systems International, Hughes LAN Systems, March 1991.

[3] K. McCloghrie, D. Perkins, and J. Schönwälder. Structure of Management Information Version 2 (SMIv2). RFC 2578, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.

[4] K. McCloghrie, D. Perkins, and J. Schönwälder. Textual Conventions for SMIv2. RFC 2579, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.

[5] K. McCloghrie, D. Perkins, and J. Schönwälder. Conformance Statements for SMIv2. RFC 2580, Cisco Systems, SNMPinfo, TU Braunschweig, April 1999.

[6] J. Schönwälder and F. Strauß. Next Generation Structure of Management Information for the Internet. In *Proc. 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, number 1700 in LNCS, pages 93–106. Springer, October 1999.

[7] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson, December 2002.

[8] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, Nortel Networks, Intel, Cisco, IPHighway, PFN, Allegro Networks, March 2001.

[9] R. Enns. NETCONF Configuration Protocol. RFC 4741, Juniper Networks, December 2006.

[10] F. Strauß and J. Schönwälder. Next Generation Structure of Management Information (SMIng) Mappings to the Simple Network Management Protocol (SNMP). RFC 3781, TU Braunschweig, IU Bremen, May 2004.

[11] F. Bak, E. Lear, and R. Droms. Procedures for Renumbering an IPv6 Network without a Flag Day. RFC 4192 (Informational), September 2005.

[12] P. Nikander, J. Kempf, and E. Nordmark. IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756 (Informational), May 2004.

[13] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461 (Draft Standard), December 1998.

[14] S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. RFC 2462 (Draft Standard), December 1998.

[15] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies, December 2002.

[16] D. Harrington and J. Schönwälder. Transport Subsystem for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-ietf-isms-tmsm-12.txt>, Huawei Technologies (USA), Jacobs University Bremen, February 2008.

[17] D. Harrington. Transport Security Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-transport-security-model-07.txt>, Huawei Technologies, November 2007.

[18] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, SSH Communications Security Corp, Cisco Systems, January 2006.

[19] D. Harrington and J. Salowey. Secure Shell Transport Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-secshell-10.txt>, Futurewei Technologies, Cisco Systems, February 2008.

[20] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414, Lucent Technologies, December 2002.

[21] R. Frye, D. Levi, S. Routhier, and B. Wijnen. Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework. RFC 3584, Vibrant Solutions, Nortel Networks, Wind River Systems, Lucent Technologies, August 2003.

[22] V. Marinov and J. Schönwälder. Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-marinov-isms-tlstm-00.txt>, Jacobs University Bremen, February 2007.

[23] A. G. Morgan. The Linux-PAM Application Developers' Guide. Technical report, November 1999.

[24] J. Schönwälder. Simple Network Management Protocol (SNMP) Context EngineID Discovery. Internet Draft (work in progress) <draft-ietf-opsawg-snmp-engineid-discovery-02.txt>, Jacobs University Bremen, February 2008.

[25] *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications*, August 2001.

[26] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, August 2001.

[27] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[28] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. in Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS), March 2002.

[29] Anni Järvelin, Antti Järvelin, and Kalervo Järvelin. s-grams: Defining generalized n-grams for information retrieval. *Information Processing and Management*, 43(4):1005–1019, 2007.

[30] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 651–660, New York, NY, USA, May 2005. ACM Press.

[31] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966.

[32] Thomas Bocek, Ela Hunt, and Burkhard Stiller. Fast similarity search in large dictionaries. Technical Report IFI-2007.02, University of Zurich, April 2007.

[33] Ubuntu linux. Website, March 2008. http://www.ubuntu.com/.

[34] Debian linux. Website, March 2008. http://www.debian.org/.

[35] Opendiameter. Website, March 2008. http://www.opendiameter.org/.

[36] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter Base Protocol. RFC 3588, Airespace, Inc., Nokia, Sun Microsystems, Inc., Cisco Systems, Inc., Ericsson, September 2003.

[37] Gnu lesser general public license. Website, March 2008. http://www.gnu.org/copyleft/lesser.html.

[38] Gnu general public license. Website, March 2008. http://www.gnu.org/copyleft/gpl.html.

[39] The adaptive communication environment. Website, March 2008. http://www.cs.wustl.edu/ schmidt/ACE.html.

[40] Daidalos. Website, March 2008. http://www.ist-daidalos.org/.

[41] Ec-gin. Website, March 2008. http://www.ec-gin.eu/.

[42] Smoothit. Website, March 2008. http://www.ec-gin.eu/.

[43] Opendiameter debian package repository. Website, March 2008. http://www.csg.uzh.ch/staff/morariu/opendiameter/.

[44] Debian new maintainers guide. Website, February 2008. http://www.debian.org/doc/manuals/maint-guide/index.en.html.

[45] Debian repository howto. Website, February 2008. http://www.isotton.com/debian/docs/repository-howto/repository-howto.html.

Appendix 1: Software packaging call one text

EMANICS Work Package 6 : Open Source Software Packaging and Tutoring

This call covers the task T6.2 in the first 9 months of phase 2 : july 2007 - march 2008 It has a overall maximum budget of 9 K Euros.

Proposal Sheet

The proposal has to be filled and sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before August 30th 12 AM. The flat funding for Packaging and Tutoring will be of 3K per initiative.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailled description of the addressed software (2-4 pages) together with a precise presentation of the packaging efforts made (2-3 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

==========

0. Proposal Title:

1. Addressed Software

name and short description of the existing Open Source project addressed

2. Detailed List of Packaging and Documentation Activity planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

3. Expected Impact

(what new markets the enhanced software will conquer, how many distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support)

Appendix 2: Software development call one text

EMANICS Work Package 6 : Open Source Software Initiatives
This call covers the first 9 months of phase 2 : july 2007 - march 2008)
It has a overall maximum budget of 45K Euros.

Proposal Sheet

The proposal has to be filled and sent to the WP Leader : Olivier Festor, Olivier.Festor@loria.fr) before August 30th 12 AM. Cooperative Open Source developments will be favored over proposals including a single participant. The budget for Open Source overall was reduced in phase 2. In order to fund several proposals, collaborative requests should not exceed 15K. Single request should not exceed 7.5K/request.

Every supported initiative commits to provide 1 months before the end of the 9 months supporting period, a detailled description of the software (5-10 pages), a precise presentation of the made changes (2-3 pages), and an impact evaluation (1-2 pages) to be included in the deliverable of the project. This is to be provided in the WP6 deliverable format which in Phase 2 will be LATEX.

Note: fill the proposal carefully and provide detailed, precise and measureable commitments. Incomplete proposals will be immediately rejected.

==========

0. Proposal Title:

1. Overall Open Source Software Description

(A 1/2 to 3/4 page description of the concerned software, its current status, its visbility, its use, ... In case of a non existing soft the emphasis should be made on its need and its impact on the community)

2. Licensing and distribution scheme

(license type and distribution scheme used for the software : GPL, LGPL, QPL, ...; available on a given forge, is it or will it be embedded in a third party software distribution, ...)

3. Detailed List of Extensions Planed under the support of EMANICS

(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact

(what new markets the enhanced software will conquer, how many

distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?

5. Cooperation level

(which parts of the extensions planed come from a cooperation among one or more partners in EMANICS, e.g. X will integrate in his software the algorithm defined by Y).

6. Cost and Requested support

(Expected cost overall + requested support) Cost includes the ressources the partner puts on the development without being supported by the NoE. These efforts must be mesurable at the end of the funding period. Request Support contains the amount of money asked to the NoE. The requested support should precisely specify how it is distributed among salary and equipment.